National University of Singapore

School of Computing

# CS3230 - Design and Analysis of Algorithms

# Final Assessment

(Semester 1 AY2024/25)

Time Allowed: 150 minutes

---

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains TWO (2) sections.
   It comprises SIXTEEN (16) printed pages, including this page.
3. This is an **Open Book** Assessment.
   The only allowed electronic device is a non-programmable calculator.
4. For Section A, use the boxes at page 11 (use 2B pencil).
   For Section B, answer **ALL** questions within the **boxed spaces** at page 12-16.
   If you leave the boxed space blank, you will get automatic free 0.5 marks.
   However, if you write at least a single character and it is totally wrong, you will get 0 mark.
   You can use either pen or pencil. Just make sure that you write **legibly**!
5. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
   Read all the questions first! Some questions might be easier than they appear.
6. You can assume that all **logarithms are in base** 2.
7. The total marks of this paper is 100 marks.
   It will then be scaled to 40% of the course weightage.

# A MCQs ($20 \times 2 = 40$ marks)

Answers: aebea ebbac edcbe dedca

1. There are two functions $f(n)$ and $g(n)$ and asymptotically $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$.
   Which of the following statements is **FALSE**?

   a). $f(n) \in o(g(n))$ Answer; should be obvious that this is the wrong one... 88% correct.

   b). $f(n) \in O(g(n))$ if $f(n) \in \Theta(g(n))$, then $f(n)$ is also in $O(g(n))$

   c). $f(n) \in \Theta(g(n))$ This is what is written

   d). $f(n) \in \Omega(g(n))$ if $f(n) \in \Theta(g(n))$, then $f(n)$ is also in $\Omega(g(n))$

   e). None of the above.

2. For any function $f$ and any number $c > 0$, define $f_c^*(n)$ as the smallest number of applications of $f$ needed to transform $n$ into a number that is at most $c$. More formally,

$$f_c^*(n) = \begin{cases} 0, & \text{if } n \le c, \\ 1 + f_c^*(f(n)), & \text{otherwise.} \end{cases}$$

   For example, if $f(n) = n/2$, then $f_1^*(n) = \lceil \log_2 n \rceil$.
   Which of the following statements is **TRUE**?

   a). If $f(n) = n^{0.1}$ and $g(n) = n^{0.9}$, then $f_7^*(n) \in o(g_7^*(n))$. Both are $\Theta(\log\log n)$.

   b). If $f(n) = n^{0.1}$ and $g(n) = n^{0.9}$, then $f_7^*(n) \in \omega(g_7^*(n))$.

   c). If $f(n) = \log n$, then $f_3^*(n) \in o(f_3^*(2^n))$. Both are $\Theta(\log^* n)$. 38% picked this.

   d). If $f(n) = \log n$, then $f_3^*(n) \in \omega(f_3^*(2^n))$.

   e). None of the above. Answer. 24% correct.

3. Consider the recurrence relation: $T(n) = T(\frac{2n}{3}) + \Theta(1)$. Which statement is **TRUE**?

   a). $T(n) \in O(1)$

   b). $T(n) \in \Theta(\log n)$ Answer; MT case 2. 88% correct.

   c). $T(n) \in \Theta(n)$

   d). $T(n) \in \omega(n)$

   e). None of the above.

4. Consider the recurrence relation: $T(n) = n^{2/3} \cdot T(n^{1/3}) + n$. Which statement is **TRUE**?

   a). $T(n) \in o(n)$ Does not feel faster than linear

   b). $T(n) \in \Theta(n)$ Unlikely to be linear

   c). $T(n) \in \Theta(n \log n)$ 35% picked this.

   d). $T(n) \in \omega(n \log n)$ Almost correct, 28% picked this.

e). None of the above. Answer: $\Theta(n \log\log n)$, as the depth of recursion is $\Theta(\log\log n)$ and the cost of each level is $\Theta(n)$. 17% correct.

5. Which of the following real-life Divide and Conquer (D&C) algorithms is an example of **case 3** of Master Theorem?

   a). Randomized Quick Select expected runtime of $T(n) = T(\frac{3n}{4}) + O(n)$ Answer: Mentioned in lecture 12 as the only one known so far (by both lecturers so far) to be analyze-able with case 3 of master theorem, so $T(n) \in \Theta(n)$. 86% correct.

   b). Binary Search, $T(n) = T(\frac{n}{2}) + O(1)$ No, it is case 2, so $T(n) \in \Theta(\log n)$

   c). Merge Sort, $T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$ No, it is case 2 again, so $T(n) \in \Theta(n \log n)$

   d). Strassen's Matrix Multiplication, $T(n) = 7 \cdot T(\frac{n}{2}) + O(n^2)$ No, it is case 1, so $T(n) \in \Theta(n^{log_2 7}) = \Theta(n^{2.807})$

   e). Karatsuba's Polynomial Multiplication, $T(n) = 3 \cdot T(\frac{n}{2}) + O(n)$ No, it is case 1 again, so $T(n) \in \Theta(n^{log_2 3}) = \Theta(n^{1.584})$.

6. There are proposals to split the term D&C into **Divide** (into *two or more* subproblems of smaller size) and Conquer and **Decrease** (into *one* subproblem of smaller size) and Conquer?
   Which D&C algorithms below is **not** of type **Decrease** and Conquer.

   a). Randomized Quick Select The single subproblem recurrence is shown in q5).

   b). Binary Search The single subproblem recurrence is shown in q5).

   c). Exponentiation: Computing $a^n \pmod m$ in $\Theta(\log(n))$ time One subproblem `https://visualgo.net/en/recursion?slide=3-3`

   d). Factorial: Computing $n!$ in $\Theta(n)$ time One subproblem `https://visualgo.net/en/recursion?slide=3-1`

   e). Karatsuba's Polynomial Multiplication Answer: The three subproblems recurrence is shown in q5). 67% correct.

7. Consider the following two assertions about a randomized procedure $\mathcal{A}$:

   **A1:** $\mathbf{E}[\text{runtime of } \mathcal{A}] < \infty$

   **A2:** $\lim_{t\to\infty} \mathbf{Pr}[\mathcal{A} \text{ does not finish by time } t] = 0$

   Which of the following statements is **TRUE**?

   a). (A1 implies A2) and (A2 implies A1). 37% picked this.

   b). (A1 implies A2) and (A2 does not imply A1). Answer: (A1 implies A2) by Markov inequality. A counterexample for the other direction: $\mathbf{Pr}[\text{runtime of } \mathcal{A} = 2^i] = 2^{-i}$ for every positive integer $i$. 24% correct.

   c). (A1 does not imply A2) and (A2 implies A1). 29% picked this.

   d). (A1 does not imply A2) and (A2 does not imply A1).

e). None of the above.

8. Place $n$ balls into $n$ bins. Each ball placement is done independently and uniformly at random. For each $i \in \{1, 2, \ldots, n\}$, let $X_i$ be the number of balls in the $i$-th bin. What is the expected value of $\sum_{i=1}^{n} \binom{X_i}{2}$?

   a). $o(n)$

   b). $\Theta(n)$ Answer: $\sum_{i=1}^{n} \binom{X_i}{2}$ is the number of pairs of balls that collide. For each pair of balls, the collision probability is $\frac{1}{n}$, so the expected number of collisions is $\binom{n}{2} \cdot \frac{1}{n} = \frac{n \times (n-1)}{2} \cdot \frac{1}{n} = \frac{n-1}{2}$. This is $\in \Theta(n)$. 35% correct.

   c). $\Theta(n \log n)$

   d). $\omega(n \log n)$

   e). None of the above.

9. Suppose the top-down Dynamic Programming (DP) $Fib(n)$ is written in this way:

```
Fib(n)
    if n <= 1, return n
    else
        A[n-1] = Fib(n-1) # original line has "if A[n-1] == None" check
        if A[n-2] == None, A[n-2] = Fib(n-2)
        return A[n-1] + A[n-2]
```

   Which of the following statements is **TRUE**?

   a). $Fib(n)$ still runs in $O(n)$ $Fib(n)$ checks $Fib(n-1)$ first and thus all $A[0..n-1]$ will be memoized anyway. When the recursion unwinds, $A[n-2]$ will always already not None. 52% correct.

   b). $Fib(n)$ runs in slow $\Omega(2^{\frac{n}{2}})$ because the `A[n-1]` case is not memoized No, see a). 40% picked this.

   c). $Fib(n)$ becomes wrong No, it still computes the $n$-th Fibonacci correctly, easy to rule out

   d). $Fib(n)$ causes runtime error No, easy to rule out

   e). None of the above Option a). is correct.

10. The goal of change-making problem is to find the minimum number of coins of denominations $d$ that add up to $n$ cents. If the denominations are $d = \{4, 3, 1, 5\}$, which value of $n$ does **not** work with the greedy strategy of picking the largest denomination that is not larger than $n$? For example, when $n = 7$, the optimal strategy is $4 + 3 = 7$ (2 coins), but the greedy strategy will pick $5 + 1 + 1 = 7$ (3 coins).

    a). $n = 1000$ Greedy $200 \times 5 = 1000$ is optimal too

    b). $n = 56$ Greedy $11 \times 5 + 1 = 56$ is optimal too

c). $n = 47$ Answer: DP $8 \times 5 + 4 + 3 = 47$; Greedy $9 \times 5 + 2 \times 1 = 47$. 96% correct.

d). $n = 38$ Greedy $7 \times 5 + 3 = 88$ is optimal too

e). $n = 19$ Greedy $3 \times 5 + 4 = 19$ is optimal too

11. The context of this MCQ is about **this** paper (read the instructions page again).
Which of the following is a good CS3230 exam taking strategy?

a). You did not prepare for this paper and plan to gamble on the MCQs and leave the entire Section B empty Actually bad, expected marks is $\frac{40}{5} + 8 \times 0.5 = 8 + 4 = 12$ out of 100 marks. But at least two students really did this in the actual paper (names omitted) and it may be their 'optimal' strategy.

b). You decide to carefully spend $\frac{150}{20} = 7.5$ minutes per MCQ for this entire Section A (MCQs) and leave the entire Section B empty Actually still not good, as there are a few easy marks in Section B. Expected marks is $X$, $X > \frac{40}{5}$ (but at most 40 if perfect MCQs, but unlikely) + 4 more. There are many more students who did not leave the exam hall throughout the 2.5 hours but their essays really mostly blanks (names omitted), so this may still be their 'optimal' strategy.

c). You almost run out of exam time for an MCQ question and decide to leave the MCQ blank This is the one that is clearly wrong, as gambling it (doable in one second) nets you an expected $\frac{1}{5} \times 2 = 0.4$ marks more for that MCQ as we do not use negative marking scheme. Interestingly, 5 students (1%) picked this. Maybe they actually gambled on this MCQ?

d). After completing the MCQs and other essays, you are very not sure about your own answer draft for one hard essay question of this paper and pray that it is marked as correct This is the opposite of e). We eventually accept this as maybe the 'very not sure' is 50-50 and leaving the answer is still a good strategy. 12% picked this.

e). After completing the MCQs and other essays, you are very not sure about your own answer draft for one hard essay question of this paper and decide to cross it out in the last minute Answer; doing this gives you a free +0.5 expected more marks as if you are very not sure, it is unlikely to be correct and we will likely arrived at the same conclusion and give you 0. 81% fully correct.

12. Consider the CD burning problem. Suppose Bob has a collection of music files that he wants to burn into CDs with the following constraints.

- Each music file cannot be split and hence cannot be burned into more than one CD.
- Each CD can contain at most two music files.

Given a set $A$ of file sizes and a positive integer $k \geq \lceil |A|/2 \rceil$, define $\mathsf{MinSizeCD}(A, k)$ as the smallest number $C$ such that if each CD has storage capacity $C$, then $k$ CDs are sufficient to store all the music files described in $A$. Let $A = \{a_1, a_2, \ldots, a_n\}$ where $a_1 \geq a_2 \geq \cdots \geq a_n > 0$. Which of the following statements is **TRUE**?

a). If $\mathsf{MinSizeCD}(A, k) = a_1$, then $k = |A|$. easy to rule out; the 1st CD only contains largest file $a_1$; but what if $a_2, a_3, \ldots a_n$ are all much smaller and can fit in pairs, then $k < |A|$.

b). If $|A|$ is even and $k = |A|/2$, then $\mathsf{MinSizeCD}(A, k) = a_1 + a_n$. a bit harder rule out; try $A = \{5, 4, 3, 1\}$, if $k = |A|/2 = 2$, then $\mathsf{MinSizeCD}(A, k) = 5 + 1 = 6$ is incorrect as it has to be $4 + 3 = 7$ so that we only use $k = 2$ CDs

c). If $\mathsf{MinSizeCD}(A, k) = a_1 + a_n$, then $k = |A|/2$. a bit harder rule out; but related to option b). we can try the same $A = \{5, 4, 3, 1\}$, if $\mathsf{MinSizeCD}(A, k) = 5 + 1 = 6$, we will need $k = 3$ CDs, not 2 CDs, as $4 + 3 = 7$ is more than 6

d). If $k > |A|/2$, then $\mathsf{MinSizeCD}(A, k) = \max\{a_1, \mathsf{MinSizeCD}(A \setminus \{a_1\}, k - 1)\}$. Answer: If $k > |A|/2$, then at least one of the $k$ CDs contains only one file. Therefore, by cut-and-paste, there exists an optimal solution where one CD contains only the largest file. 51% correct.

e). None of the above. If students ruled out a)., b)., c). and yet did not realize that d). is correct, they will pick this...

13. Consider a variant of the binary counter where a number is represented as a decimal number (base-10 integer) and not a binary number (base-2 integer). Changing a digit (a symbol in $\{0, 1, 2, \ldots, 9\}$) still costs one unit. Start from zero and perform $n$ increments ($+1$). What are the amortized and worst-case costs per increment?

a). Amortized cost: $\Theta(\log n)$, worst-case cost: $\Theta(\log n)$.

b). Amortized cost: $\omega(1)$ and $o(\log n)$, worst-case cost: $\Theta(\log n)$.

c). Amortized cost: $\Theta(1)$, worst-case cost: $\Theta(\log n)$. Answer: The same as binary counter. Changing the base does not change much. 72% correct.

d). Amortized cost: $\Theta(1)$, worst-case cost: $\omega(1)$ and $o(\log n)$. Some students may accidentally pick d). due to the $\omega(1)$ and $o(\log n)$ option...

e). None of the above.

14. Consider a variant of the binary counter where not only increment ($+1$) but also decrement ($-1$) is allowed. To ensure that the number never goes below zero, we assume that if the current number is already zero, the decrement does not change the number. Start from zero and perform any $n$ operations (increments or decrements). Which of the following statements is **TRUE**?

a). For some choices of $n$ operations, the worst-case cost per operation can be $\omega(\log n)$. Easy to rule out, cannot be more than $\log n$

b). For some choices of $n$ operations, the amortized cost per operation can be $\Omega(\log n)$. Answer: If we allow both $+1$ and $-1$, the amortized cost can be $\Theta(\log n)$. But then since there is no $\Theta(\log n)$ option, we need to pick $\Omega(\log n)$ option. 36% correct.

c). The amortized cost per operation is still $\Theta(1)$ for all choices of $n$ operations. Easy to rule out: 7 to 8 to 7 to 8 is flip-flopping situation, but 34% picked this.

d). While the amortized cost per operation is still $o(\log n)$ for all choices of $n$ operations, the amortized cost per operation can be $\omega(1)$ for some choices of $n$ operations. this is as tricky as 13.d). above...

e). None of the above.

15. Suppose there is a reduction from problem $A$ to problem $B$ satisfying the following property

- In $O(n^3)$ time, an instance $I_A$ of size $n$ for problem $A$ can be reduced to an instance $I_B$ of size $2n$ for problem $B$ such that the following statement holds for $(I_A, I_B)$:
  - Given a solution for $I_B$, a solution for $I_A$ can be computed in $O(n^3)$ time.

Which of the following statements is **TRUE**?

a). If $B$ is solvable in $O(n!)$ time, then $A$ is solvable in $O(n!)$ time. Should be $O((2n)!)$.

b). If $B$ is solvable in $O(2^n)$ time, then $A$ is solvable in $O(2^n)$ time. Should be $O(2^{2n})$.

c). If $A$ is solvable in $O(n!)$ time, then $B$ is solvable in $O(n!)$ time. Info on $A$ does not mean anything for $B$

d). If $A$ is solvable in $O(2^n)$ time, then $B$ is solvable in $O(2^n)$ time. Similar as c). easy to rule out.

e). None of the above. Answer. 52% correct.

16. Given any decision problems $A$ and $B$ with $A \leq_P B$ (Karp reduction).
Which of the following statements is **TRUE**?

a). ($A \in \mathsf{P}$ implies $B \in \mathsf{P}$) and ($A \in \mathsf{NP}$ implies $B \in \mathsf{NP}$). ($A \in \mathsf{P}$ implies $B \in \mathsf{P}$) should be quickly ruled out

b). ($A \in \mathsf{P}$ implies $B \in \mathsf{P}$) and ($B \in \mathsf{NP}$ implies $A \in \mathsf{NP}$). Same as above

c). ($B \in \mathsf{P}$ implies $A \in \mathsf{P}$) and ($A \in \mathsf{NP}$ implies $B \in \mathsf{NP}$). 47% picked this.

d). ($B \in \mathsf{P}$ implies $A \in \mathsf{P}$) and ($B \in \mathsf{NP}$ implies $A \in \mathsf{NP}$). Answer: To see that ($B \in \mathsf{NP}$ implies $A \in \mathsf{NP}$), observe that Karp reduction allows us to get a certificate for $A$ that can be verified in polynomial time: Just do a reduction to $B$ and take the certificate for the $B$-instance. 35% correct.

e). None of the above.

17. Consider the three complexity classes for decision problems: P, NP, and NP-complete.
Which of the following statements is **FALSE**?

a). If $\mathsf{P} \cap \mathsf{NP\text{-}complete} \neq \emptyset$, then $\mathsf{P} = \mathsf{NP}$. We show this statement to be TRUE in lec10, so hopefully students rule this out quickly.

b). For all $A \in \mathsf{P}$ and for all $B \in \mathsf{NP\text{-}complete}$, $A \leq_P B$ (Karp reduction). Not many students will see this... Indeed 43% picked this.

c). $\mathsf{NP\text{-}complete} \neq \emptyset$. We show at least one C-SAT, then reduce to many more in lec10, students should rule this out...

d). $\mathsf{P} \subseteq \mathsf{NP}$. We show this statement to be TRUE in lec10, so hopefully students rule this out quickly.

e). None of the above. Answer. 34% correct.

18. Which of the following sorting algorithms is the best for sorting $n = 10^9$ (1 billion) 64-bit signed integers? Assume that your computer has 1 Gigabytes of RAM (i.e., it can store an array of $\approx 10^8$ 64-bit signed integers in RAM).

a). Insertion Sort Obviously not, $O(n^2)$

b). Merge Sort Still $\Theta(n \log n)$

c). Radix Sort with one pass of Counting Sort with $k = 2^{64}$ This is one pass of very gigantic counting sort, ran out of memory, $2^{64} > 10^8$

d). Radix Sort with four passes of Counting Sorts with $k = 2^{16}$ Answer; this time the options c). and e). are clearly inferior to this one. 66% correct.

e). Radix Sort with twenty passes of Counting Sorts with $k = 10$ (base 10/Decimal) This is 20 passes of base-10 standard counting sorts, not the best

19. *Dynamic* selection problem is the problem of finding the $i$-th smallest element of a data structure where new elements can be added and existing elements can be deleted or updated.
Assume that the data structure is initially empty.
Which of the following strategy solves the dynamic selection problem in $o(n)$ time per operation?

a). This is not possible: the lower bound of selection is at least $n$ steps Only if the underlying data structure is an unsorted array. 26% picked this.

b). Re-sort the array using insertion sort per each add/delete/update, then report index $i$ Each insertion sort requires $O(n)$ (delete is also $O(n)$), thus not $o(n)$, despite only $\Theta(1)$ to report the $i$-th index...

c). Use Order Statistics Tree This is what is mentioned at the back of lec12. Use Order Statistics Tree: balanced BST augmented with size of subtree information. Then, we can solve find min and find max in $\Theta(1)$ and select and rank in $O(\log n)$, while all add/delete/update can all be done in $O(\log n)$, thus satisfying the $o(n)$ requirement. 45% correct.

d). Use Quickselect after each add/delete/update It runs in expected linear-time each, so it not $o(n)$

e). Use median of medians selection algorithm after each add/delete/update It runs in worst-case linear-time each, so it is not $o(n)$

20. Consider a directed weighted graph $G = (V, E)$ and a source vertex $s$.
Which of the following statements about shortest paths on $G$ from $s$ is **FALSE**?

a). Given positive weighted graph, Dijkstra's can fail to produce the shortest paths From lec03, Dijkstra's will be always correct, we had the proof of correctness. 84% correct.

b). Given non-negative weighted graph, BFS algorithm can fail to produce the shortest paths From CS2040S

c). If the graph has negative-weight cycle reachable from $s$, the shortest paths are ill defined Yes

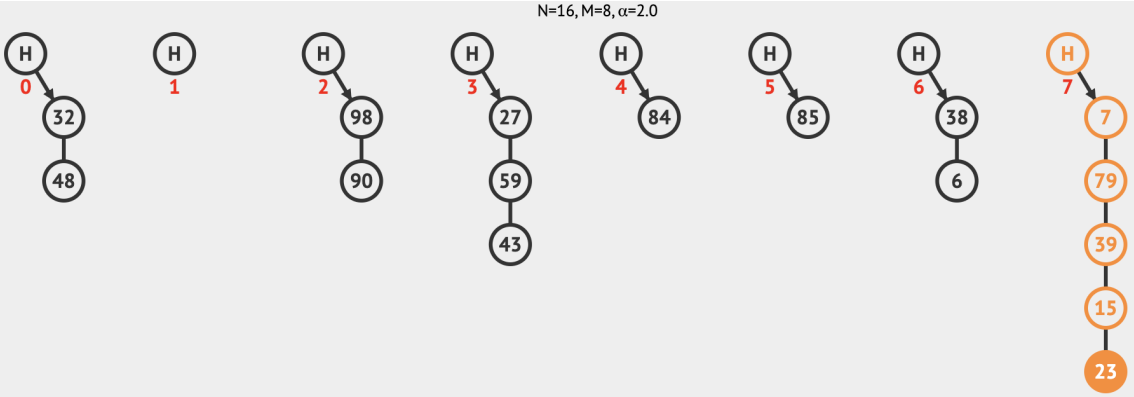
d). Every subpath of a shortest path is also a shortest path Yes

e). Given negative weighted graph, Dijkstra's can fail to produce the shortest paths Yes

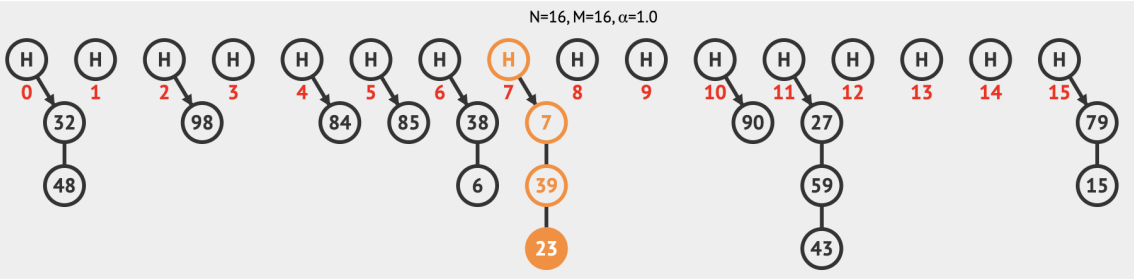
# B Essay

## B.1 Hash Table Rehash Amortized Analysis (15 marks)

Hash table can be implemented in several ways. Below, we show Separate Chaining implementation. The search performance of such hash table is influenced by the load factor $\alpha = \frac{N}{M}$ (which is also the expected chain length) where $N$ is the number of keys and $M$ is the current table size. For example, given the following hash table with $\alpha = \frac{16}{8} = 2$ (the expected chain length is 2), searching for element 23 can take up to 5 steps as it is contained at a chain/list of length 5 (all 5 of those keys are hashed to the same base address 7 via $h(v) = v\%M$ hash function).




If all of these $N = 16$ keys are rehashed into a **twice bigger** hash table of size $M = 16$, then $\alpha$ decreases to $\alpha = \frac{16}{16} = 1$ and as we can see, the expected chain length is now 1.




A strategy to keep $\alpha$ small is to **double** the hash table size $M$ and rehash the existing $N$ keys if the next insertion causes $\alpha > 2$. A strategy to keep $M$ small is to **halve** the hash table size $M$ and rehash the existing $N$ keys if the next deletion causes $\alpha < \frac{1}{2}$. However, each of rehashing is an expensive $O(N)$ operation. Show that the amortized cost of insertion and deletion are still $O(1)$ even if we rehash all $N$ keys when needed. You can use either aggregate, accounting, or potential method.

**Clarification:** As discussed in the class, under reasonable assumptions, it is possible to show that the expected cost of both insertion and deletion is at most $\alpha + 1$, where $\alpha$ is the load factor at the beginning of the operation. As randomness is not the focus of this problem, in your solution, you may assume (without a proof) that the cost of both insertion and deletion is at most $\alpha + 1$, which is always $O(1)$, as $\alpha$ is upper bounded by a constant by the design of the data structure. Also, the hash table is initially empty ($N = 0, M = 1$). Hash table size $M$ will never drop to 0.

## 11 mark solutions (insert only)

Actual cost of insert (without rehash): \$1
Actual cost of insert (with rehash): $\$(N + 1)$ ($\$N$ also accepted)

### Aggregate method

Aggregate is the same as in lec08 (with a very small edit, as at the second insertion, $\alpha = \frac{2}{1} = 2$ do not trigger rehash yet – this is negligible). Cost of $N$ insertions $T(N)$ is thus:

$T(N) = \sum j = 1^{\log(N-1)} 2^j + \sum i = 1^N 1$
$T(N) \leq 2 \cdot (N - 2) + N$ (the one in lec8 is $\leq 2 \cdot (N - 1) + N$)
$T(N) \leq 3 \cdot N$

### Accounting method

For each insertion that does not cause a rehash, let the cost be \$3, we put \$2 in the bank and use \$1 for the actual cost. Amortized cost is \$3.

When rehash occurs, note that $N/2$ elements have been inserted since the last rehash, and therefore we have $N/2 \times \$2 = \$N$ to pay for the expensive rehash, which costs $\$N + 1$. So, the amortized cost is 1. (Note: many students claim that all $N$ elements have money in them. This is not true as the money from the first $N/2$ elements are used up in the previous rehash. 8 points for students who make this claim.)

### Potential method

Define $\phi = 2(N - M) + 2$ (we also accept $2(N - M)$, although this is technically wrong since it violates $\phi(0) = 0$, recall that at the start of the paper, we clarify that $N = 0, M = 1$). When we insert without rehash, $N$ increases by 1, $M$ does not change, so $\Delta\phi = +2$. Amortized cost = 1+2 = 3.

When we insert and rehash, $N$ increases by 1, while $M$ doubles from $N/2$ (initially $\alpha = N/M = 2 \Rightarrow M = N/2$ to $N$. $\phi(i - 1) = N$ and $\phi(i) = 2$. Therefore, $\Delta\phi = 2 - N$, so that the amortized cost is $(N + 1) + (2 - N) = 3$.

Potential never goes negative as after every rehash, $N > M$ (by right you should mention this, but we do not penalize if you miss this out).

**15 mark solutions:**

Actual cost of insert (without rehash): \$1
Actual cost of insert (with rehash): $\$N + 1$ ($\$N$ also accepted)

**Accounting method**

For each insertion that does not cause a rehash, let the cost be \$3, we put \$2 in the bank and use \$1 for the actual cost. Amortized cost is \$3. For each deletion that does not cause a rehash, let cost be \$2, we put \$1 in the bank and use \$1 for the actual cost, amortized cost is \$2.

Note that at the end of each doubling rehash, we will have $N = M + 1$, and at the end of each halving rehash, we will have $N = M - 1$.

Doubling rehash: In either case, between the previous rehash and the next rehash, at least $N$ items would be inserted ($N$ items also accepted) from the previous rehash (Note: at least because deletions could have occurred between the inserts), and therefore we would have banked $N \cdot \$2 = \$2N$ which would pay for the rehash cost of $\$2N$ (since we now have $\$2N$ items).

Halving rehash: We would have deleted at least $N/2$ items since the last rehash. Since we bank \$1 for each deletion, we would have banked $N/2 \cdot \$2 = \$N$ whioch would pay for the rehash cost of $\$N/2$ (since we now have $N/2$ items).

Note: some students only bank money on insertions and not deletions, claiming that deletions are free as they can use the money from insertions, and the halving rehash can use the bank balance from deletions. This is not true as there can be up to $O(\log n)$ halving deletions in a row without any insertions, and we would run out of bank balance.

**Potential method**

Define $\phi = 2|N - M| + 2$. When rehash does not occur, $\Delta\phi \leq 2$ since $M$ does not change and $N$ changes by 1. Amortized cost $\leq 2 + 1 = 3$.

Doubling rehash: before rehash, $M = N/2$ and after rehash, $M = N$, and $N$ increases by 1. $\phi(i-1) = N$ and $\phi(i) = \$2$, so $\Delta\phi = 2 - N$. So amortized cost is $(2 - N) + (N + 1) = 3$.

Halving rehash: before rehash, $M = 2N$ and after rehash, $M = N$. $N$ decreases by 1. $\phi(i-1) = N$ and $\phi(i) = 2$, so $\Delta\phi = 2 - N$. Again, amortized cost is $(2 - N) + (N + 1) = 3$.

Grading remarks: The average score for all 430 students is 6.694/15.0.

## B.2 Activity Selection Problem (15 marks)

We are given a set of activities $A = \{a_1, a_2, \ldots, a_n\}$, where activity $a_i$ takes place during the time interval $[s_i, f_i)$, with $s_i < f_i$.

- We say that two activities $a_i$ and $a_j$ overlap if $[s_i, f_i) \cap [s_j, f_j) \neq \emptyset$, meaning that their time intervals overlap.
  - We emphasize that each activity $a_i$ overlaps with itself $a_i$.

- We say that a subset $D \subseteq A$ of activities is dominating if for every activity $a_i \in A \setminus D$, there exists an activity $a_j \in D$ such that $a_i$ and $a_j$ overlap.

We aim to compute a smallest dominating subset $D \subseteq A$ of activities.

### B.2.1 Incorrect Attempt (6 marks)

Show that the following greedy algorithm is incorrect:

- $D \leftarrow \emptyset$.
- $B \leftarrow A$.
- While $B \neq \emptyset$, do the following steps.
  - Select an activity $a \in A \setminus D$ to maximize the number of activities in $B$ overlapping with $a$.
  - Add $a$ to $D$.
  - For each activity $b \in B$ overlapping with $a$, remove $b$ from $B$.
- Return $D$.

Intuitively, this greedy algorithm aims to maximize the progress in each iteration. For example, the output of the algorithm on the input

$$A = \{[1,2), [2,3), [3,4), [4,5), [5,6), [1,3), [1,4), [3,6)\}$$

is $D = \{[1,4), [3,6)\}$, which is optimal. The algorithm selects $a = [1,4)$ in the first iteration. After that, we have $D = \{[1,4)\}$ and $B = \{[4,5), [5,6)\}$. The algorithm selects $a = [3,6)$ in the second iteration. After that, we have $D = \{[1,4), [3,6)\}$ and $B = \emptyset$.

A possible counterexample: $A = \{[1,2), [1,3), [2,3), [2,4), [3,4), [3,5), [4,5)\}$.
Optimal solution: $D = \{[1,3), [3,5)\}$.
The solution produced by the greedy algorithm: first selecting $[2,4)$, and then selecting one of $\{[1,2), [1,3)\}$ and one of $\{[3,5), [4,5)\}$.

Grading remarks: The average score for all 430 students is 2.115/6.0.

The most common mistake (roughly 70 students): the algorithm selects from $A \setminus D$ and not $B$, so the activities that already overlap with the current $D$ are still under consideration and can be added to $D$. In particular, the algorithm is correct on the example given in the exam sheet.

Other mistakes:

- The optimization problem is minimizing and not maximizing.
- The size of the solution is measured by the number of activities and not the sum of lengths over all activities.
- Trying to give a proof without providing a counterexample.

### B.2.2 Correct Algorithm (9 marks)

Design a polynomial-time algorithm that outputs a smallest dominating subset of activities.
Proof of correctness and time complexity analysis are not required for this question.

**Hint:** Modify the incorrect greedy algorithm above by using a different way to select $a \in A \setminus D$.

We use the same algorithm with a different greedy choice of $a \in A \setminus D$:

- Select $a_i = [s_i, f_i) \in B$ to minimize the finish time $f_i$.
- Let $S \subseteq A \setminus D$ be the set of all activities in $A \setminus D$ overlapping with $a_i$.
- Select $a_j = [s_j, f_j) \in S$ to maximize the finish time $f_j$.
- Our greedy choice is $a = a_j$.

The new algorithm can be implemented to finish in polynomial time. To show that the algorithm is correct, we just need to argue that there exists an optimal solution that contains our greedy choice $a_j$.

Observe that in any optimal solution, at least one activity $a_k = [s_k, f_k) \in S$ must be included, since otherwise $a_i$ is not dominated. If $a_k = a_j$, then we are done. Otherwise, we just need to show that replacing $a_k$ with $a_j$ does not make the solution non-dominating. In other words, we want to show that there is no activity $a_l = [s_l, f_l) \in B$ that overlaps with $a_k$ and does not overlap with $a_j$.

Suppose such an activity $a_l = [s_l, f_l) \in B$ exists. Since $a_l$ does not overlap with $a_j$, there are two possibilities.

- Case 1: $f_l \leq s_j$. Our choice of $a_j$ guarantees that $s_j < f_i$, so $f_l < f_i$, contradicting our choice of $a_i$.
- Case 2: $f_j \leq s_l$. Our choice of $a_j$ guarantees that $f_k \leq f_j$, so $f_k \leq s_l$, meaning that $a_l$ also does not overlap with $a_k$, contradicting our choice of $a_l$.

Since both possibilities are impossible, such an activity $a_l$ does not exist.

Grading remarks: The average score for all 430 students is 1.050/9.0.

Some mistakes:

- Any attempt to modify the algorithm with a tie-breaking rule must be incorrect.
- Select the longest one does not work (even with some tie-breaking rules).
- Select the longest one does not work (even with some tie-breaking rules).
- Select the one that starts/ends first/latest does not work (even with some tie-breaking rules).
- Minimum dominating set and maximum independent set (i.e., the set of activities that are mutually compatible) are very different things (e.g., a minimum dominating set might not be an independent set).

Ambiguity in algorithm description:

- Find $a$ in $A \setminus D$ that maximizes (...) and minimizes (...) - which first?
- First/last activity - sorted ordering according to what? Start time or end time or others?

My preferred solution:

- Select $x$ in $B$ that finishes first (*).
- Among all activities in $A \setminus D$ that overlaps with $x$, select a to be the one that maximizes the finish time (**).

There are some flexibilities:

- (*) can be replaced with "starts first"
- (**) can be replaced with "maximizes the number of activities in $B$ overlapping with $a$"
- However, you cannot simultaneously do these two modifications! (I consider this as a minor mistake and still give full mark)

Alternative (equivalent) solutions:

- Select $a$ in $A \setminus D$ to maximize $x$ such that a overlaps with all activities in $B$ with start time $\leq x$.
- Select $a$ in $A \setminus D$ to maximize $x$ such that $a$ overlaps with the first $x$ activities in $B$, where the activities in $B$ are sorted according to start time.

### B.3 `3-Clique-Cover` (20 marks)

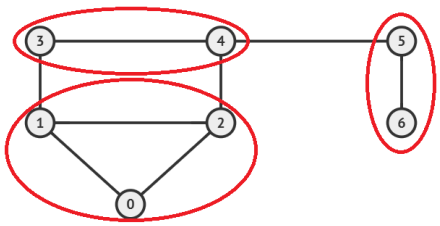
Given an **undirected, unweighted, and connected** graph $G = (V, E)$ with at least 2 vertices, let $S$ be the set of (all possible) cliques in $G$. `3-Clique-Cover` is a decision problem to decide whether it is possible to choose **at most 3 cliques** from $S$ such that every vertex in $V$ appears in **exactly one of the cliques**.

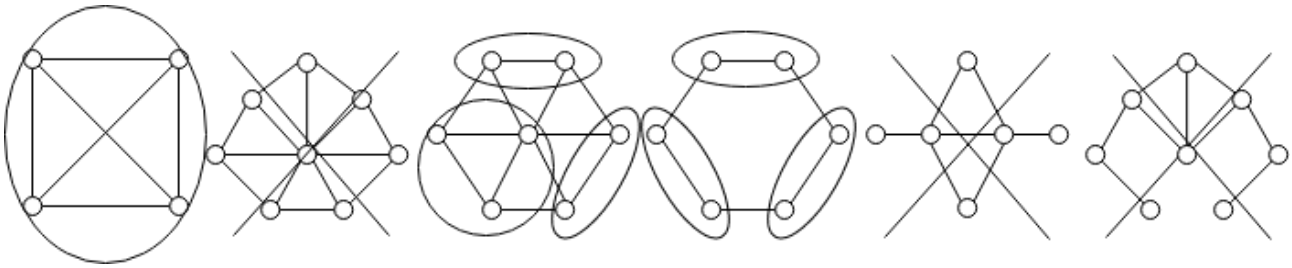
We also define the **distance** between 2 distinct vertices in $G$ as the number of edges in the (unweighted) shortest path connecting the 2 vertices. The diameter of a graph, $diam(G)$, is then the **maximum distance** between any 2 vertices in $G$.

#### B.3.1 Manual Test Cases (6 marks)

Show your understanding of the definition of $diam(G)$ and `3-Clique-Cover` above on the following graphs (see box B.3.1 of the Answer Sheet). For each graph, state the diameter of the graph and whether it can be covered by **at most 3 cliques**. If it can be covered, please circle the cliques used to cover the graph. Each correct answer worth 1 mark. For the example graph $G$ below, the answer is: ($diam(G) = 4$, YES) and the 3 cliques used to cover $G$ are circled. Note that if the answer is a NO, you need to clearly state it, i.e., ($diam(G)$, NO) .

(1, YES), (2, NO), (2, YES), (3, YES), (3, NO), (6, NO).



These graphs have been specially crafted as hint for the answers for later parts.
For leftmost $K_4$ ($diam(G) = 1$), cover it with 1 $K_4$ clique (for B.3.4.1 later) or with 2 or 3 cliques.
The rightmost graph has $diam(G) = 6$, one cannot cover this with just 3 cliques (for B.3.4.2 later).

Grading remarks: a few students forgot to compute diameters and/or answer YES but not showing the certificate. The required answer format is ($diam(G)$ =?, YES-with certificate/NO). Both the computation of $diam(G)$ AND the YES (with certificate) or NO decision have to be correct to score 1 mark for that test case). The average score for all 430 students is 4.595/6.0.

### B.3.2 Prove `3-Clique-Cover` is in NP (3 marks)

Prove that `3-Clique-Cover` is in NP.

Given a certificate for a YES-instance of `3-Clique-Cover`,
i.e., the 1, 2, or up to 3 cliques (graph 1, 3, 4 at B.3.1), of polynomial size,
we can verify the certificate in polynomial time:
a). whether the number of cliques that we are given is $\leq 3$, doable in $O(1)$,
b). whether the given cliques are really cliques, up to $O(E)$ or $O(V^2)$, and,
c). all the $V$ vertices are covered exactly once by those 1, 2, up to 3 cliques, doable in $O(V)$.
PS: there can be differences in implementation details/assumptions on how the checks can be done, so any *polynomial* verification strategy is accepted.

Grading remarks: a substantial number of students forgot one the parts above, e.g., assuming the certificates are really 3 cliques without checking or assuming that union of all cliques equals to $V$ is enough (not really enough to detect if each vertex is covered by just one clique, one should also test if the intersection of all cliques is empty) (given 2 marks). It is rare for student (who answer this question) to miss 2 checks (given only 1 mark). PS: 48/430 students (11%) still leave this blank despite repeated reminders not to leave this easy question blank. The average score for all 430 students is 2.141/3.0.

### B.3.3 Prove `3-Clique-Cover` is NP-complete (6 marks)

Prove that `3-Clique-Cover` is NP-complete by reduction from `3-Coloring`, which asks whether the vertices of a graph $G = (V, E)$ can be colored using at most 3 colors such that no two adjacent vertices have the same color. You are told that the `3-Coloring` problem is NP-complete.

Hopefully this is easy enough and not skipped by more than half of CS3230 S1 AY24/25 students... Prof Halim shown CLIQUE $\leq_p$ VC during the optional lecture on wk11b in order to show this reduction idea of complement graph $E'$... Given any $G = (V, E)$ instance of `3-Coloring`, we construct an instance of `3-Clique-Cover` on complement (this is the important keyword) graph $G' = (V, E')$ with the same set of vertices $V$ as $G$ but now $E'$ is as follows: two vertices are adjacent in $G'$ if and only if they are not adjacent in $G$. Clearly this reduction runs in polynomial time (in $O(V^2)$).

Prove that a YES-instance of `3-Coloring` $\rightarrow$ YES-instance of `3-Clique-Cover`. Now, if $G$ is a YES-instance of `3-Coloring`, the vertices of the same color in $G$ will form a clique in $G'$ since no two vertices (with the same color) are adjacent in $G$ (and they are all now adjacent in $G'$ – a clique). Since we can color $G$ with at most 3 colors, therefore we will be able to cover $G'$ with at most 3 cliques, a YES-instance of `3-Clique-Cover`. Proven.

Prove that a YES-instance `3-Coloring` $\leftarrow$ YES-instance of `3-Clique-Cover`. Now, if $G'$ is a YES-instance of `3-Clique-Cover`, there are up to 3 cliques that can be used to cover all vertices in $G'$, we can color up to those 3 cliques with 3 colours, e.g., RED, GREEN, and BLUE. Now, at $G$, any two vertices with the same color will not have an edge in $E$, a YES-instance of `3-Coloring`. Proven.

We have shown that `3-Coloring` $\leq_p$ `3-Clique-Cover`. Combined with the earlier section that `3-Clique-Cover` is in NP, we conclude that `3-Clique-Cover` is NP-complete.

Grading remarks: 297 students (69%) still left this blank, i.e., we are still unable to make this less than half for the last three semesters... Among those who attempt, majority are correct as long as they mention complement (or 'similar terms': inverse, edges not in $G = (V, E)$, etc). A common mistake is to include 'color' in the (polynomial-time) reduction, e.g., Given a YES-instance $G = (V, E)$ instance of `3-Coloring` and it's colours..., group vertices of the same colour into one clique... The colours are not part of the input of `3-Coloring`. The average score for all 430 students is x.???/6.0.

### B.3.4 Solve Two Special Cases (5 marks)

Design the best algorithms to solve `3-Clique-Cover` on $G$ with the following diameters:
(the given graph $G$ is guaranteed to have the stated $diam(G)$, you do not have to check for this again).
Prove the correctness of your algorithms.
Finally, analyze the time complexities of your algorithms using $\Theta$ notation.

1. $diam(G) = 1$ (2 marks)

   Simply always output YES, $\Theta(1)$ (0.5+0.5 marks).

Proof of correctness: The certificate is $G$ itself. $diam(G) = 1$ means that the input graph itself is already a clique and we can always cover it with just one clique, $G$ itself (1 mark).

Grading remarks: 155 students (36%) still leave this blank. Among those who try, almost everyone are correct. The average score for all 430 students is 1.296/2.0.

2. $diam(G) \geq 6$ (3 marks)

   Simply always output NO, $\Theta(1)$ too (0.5+0.5 marks again).

   The proof of correctness is a bit long (2 marks): Let $diam(G) \geq 6$. Let $u$ and $v$ be the two vertices in $G$ with the furthest distance, i.e., along the extremities of the diameter, then this furthest distance is $\geq 6$. Then a shortest path $P$ between $u$ and $v$ has $\geq 6$ edges and thus involving $\geq 7$ vertices including $u$ and $v$.

   Suppose there are $\geq 3$ of the vertices in the shortest path $P$ are in the same clique, then $\geq 2$ of them are not adjacent along $P$ so the edge between them will form a 'shortcut' along $P$, so then $P$ cannot be the shortest path. Contradiction. Hence, $\leq 2$ vertices in the shortest path $P$ can be in the same clique (of size $K_2$, an edge).

   Therefore, it is impossible to cover $P$ that has $\geq 7$ vertices with $\leq 3$ cliques since we can only cover at most $3 \times 2 = 6$ vertices with three edges (cliques of size $K_2$).

   Another way of seeing this: the maximum diameter involving 3 cliques (with $diam(G) = 1$) and 2 edges connecting these 3 cliques is 5.

   Note that it is wrong to say there cannot be any other edges adjacent to the vertices in $P$, look at the last/rightmost graph in Subsection B.3.1.

   It is also wrong to say that if a $P_7$ (Path of 7 vertices) is a subgraph (not the path along the diameter) then it is not possible to cover in at most 3 cliques, consider $K_7$ for example.

   Grading remarks: 312 students (73%) still leave this blank. Among those who try, majority are correct and understand the implication of $diam(G) \geq 6$ as above. The average score for all 430 students is 1.030/3.0.

### B.4 Computing Diameter of a Directed Weighted Graph (10 marks)

The definitions in this question are similar *but slightly different* from the previous Section B.3.

Given a **directed, weighted (ones, any positive weights, zeros, or negative weights), and not necessarily connected** graph $G = (V, E)$ with at least 2 vertices, we define the **distance** between 2 distinct vertices in $G$ as the path weight in the **weighted** shortest path connecting the 2 vertices. The diameter of a graph, $diam(G)$, is the **maximum distance** between any 2 vertices in $G$,

which can be $\infty$ if $G$ is disconnected.

**Additional constraints:**

- The number of edges in the graph satisfies $|E| \in \Theta(|V|^{1.5})$.
- Also, there is **no negative weight cycle** in the graphs.

Now, design an algorithm to compute $diam(G)$.
You may use any algorithm(s) that you have learned in class.

Diameter of a graph $G = (V, E)$, i.e., $diam(G)$, is simply the longest of all shortest paths from all possible source vertices in $G$ (there are $V$ possible sources). So this computation of $diam(G)$ requires us to solve the All-Pairs (not just Single-Source) Shortest Paths (APSP) problem first. The APSP computation can be done in many ways and is the bottleneck (see below).

Finally, after we computed the APSP information, we can find $diam(G)$ by trying a simple $\Theta(V^2)$ brute force $u, v \in V$ and find the largest $APSP[u][v]$. This $\Theta(V^2)$ is dominated by any other APSP choices below:

- Wrong answer is easy, 0 mark. There is a loophole that has been clearly closed, i.e, the actual original answer of Section B.3, i.e., to compute APSP using $V$ calls of $O(V + E)$ BFS, as the graphs over there are unweighted. This is in $O(VE) = O(V^{2.5})$ for dense graph with $E \in \Theta(V^{1.5})$ but only works for unweighted graphs. As per clearly given marking scheme, this will be marked down as 0.
- Blank is also easy, 0.5 marks, but we are not expecting that many blanks for this last question, hopefully at most just $\frac{1}{4}$ leave this blank.
- $\omega(V^{2.5})$ but only for general non-negative weighted graphs, is $V$ calls of $O((V + E) \log V)$ Dijkstra's implemented with Binary Heap, thus $O(VE \log V) = O(V^{2.5} \log V)$ for dense graph with $E \in \Theta(V^{1.5})$.
- $O(VE)$ but only for general non-negative weighted graphs, is $V$ calls of $O(V \log V + E)$ Dijkstra's implemented with Fibonacci Heap (although we never study Fibonacci heap in details, not in CS2040S, not in CS3230, this specific time complexity is mentioned in passing in lec3 by Prof Yi-Jun and repeated on purpose by Prof Halim on lec13), thus $O(V^2 \log V + VE)$. This time complexity is dominated by $O(VE) = O(V^{2.5})$ for dense graph with $E \in \Theta(V^{1.5})$. So, just by changing Binary Heap to Fibonacci Heap, students will get 7 out of 10 marks.
- $\omega(V^{2.5})$ is $V$ calls of $O(VE)$ Bellman-Ford (mentioned in lec13), which is $O(V^2E) = O(V^{3.5})$ for dense graph with $E \in \Theta(V^{1.5})$ (we do not expect many students to do this); Another $\omega(V^3)$ solution is the Section 23.1 of CLRS (not many students will know this), the $\Theta(V^4)$ version of SLOW-APSP, which is to raise Adjacency Matrix $W$ of the graph into its $(V - 1)$-th power (as the longest shortest path can have up to $V - 1$ edges), or $W^{n-1}$ (this is actually DP). The $\Theta(V^3 \log V)$ version FASTER-APSP of Section 23.1 of CLRS, which is to compute $W^{n-1}$ using

lec03 DnC matrix exponentiation, is also unlikely to be known by students. Some students who are aware that doing $V$ calls of Bellman-Ford (lec13) worth more than $V$ calls of Dijkstra's (from lec13) for this problem, will get this 8 out of 10 marks.

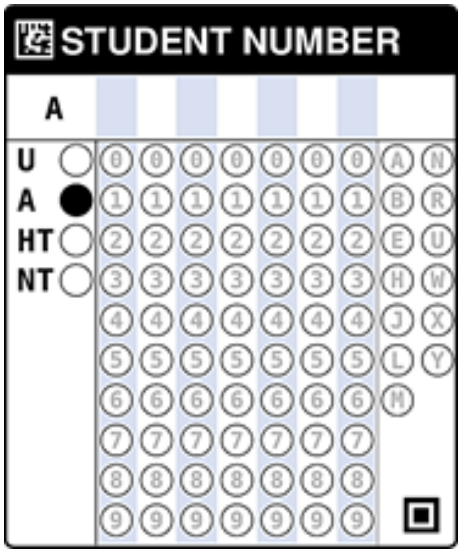
- $\Theta(V^3)$ for weighted graphs, is the famous three-nested loops $\Theta(V^3)$ Floyd-Warshall algorithm. Easy 9 out of 10 marks for students who are aware of this extra algorithms (the name of this algorithm was purposely mentioned in passing in lec13).
- $O(VE)$ for weighted graphs, is the not-so-famous Johnson's algorithm that calls $O(VE)$ Bellman-Ford first (discussed in lec13) and if there is negative weight edges; to re-weight the edge weights accordingly; before running the $V$ calls of Dijkstra's implemented with Fibonacci-heap discussed earlier. The overall time complexity is thus $O(V^2 \log V + VE)$ again and is dominated by $O(VE) = O(V^{2.5}$ for dense graph with $E \in \Theta(V^{1.5})$. This last 1 mark is reserved for anyone who knows this, otherwise the maximum mark of this paper is 99 marks. Although the name 'Johnson's algorithm' (especially the reweighting part) is never mentioned in class, its two constituent sub-routines: Bellman-Ford and Dijkstra's (with Fibonacci heap) are indeed mentioned.

Grading remarks:

- out of 430:
  - blank - 248
  - Dijkstra's with Fibonacci heap - 5
  - Dijkstra's without Fibonacci heap - 52 (note: many did not mention using Fibonacci heap or runtime, so those answers are placed them here by default. also there were quite a number who stated Dijkstra's runs in $O(V + E)$ - as this is incorrect, such answers are also placed here)
  - Bellman-Ford - 47
  - Floyd-Warshall - 27
  - Johnson's - 1
- No one attempted methods using like matrix multiplication
- Most common mistakes were trying to find max distance, or using something like MST to try and obtain diameter
- 1 mark is deducted for issues like not correctly stating how to obtain diameter from the computed All-Pairs Shortest Paths (extremely rare), ambiguous notions where it was not clear the student was indeed running SSSP once for each vertex as source to obtain APSP.
- We did not mark down when students did not write runtime, or stated some runtimes incorrectly, except for the above Dijkstra's case (as without mentioning Fibonacci heap, we will mark them down by default).

# The Answer Sheet

Write your Student Number in the box below using **(2B) pencil**.
**Do NOT write your name**.




**Write your MCQ answers in the special MCQ answer box below for automatic grading.**
We do not manually check your answer.
Shade your answer properly (use (2B) pencil, fully enclose the circle; select just one circle).

| No | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| A | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| B | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| C | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| D | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| E | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |

| No | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| A | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| B | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| C | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| D | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |
| E | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ | ◯ |

Box B.1. Hash Table Rehash Amortized Analysis

**Grading scheme:** Your answer will be graded using the special marking scheme below.

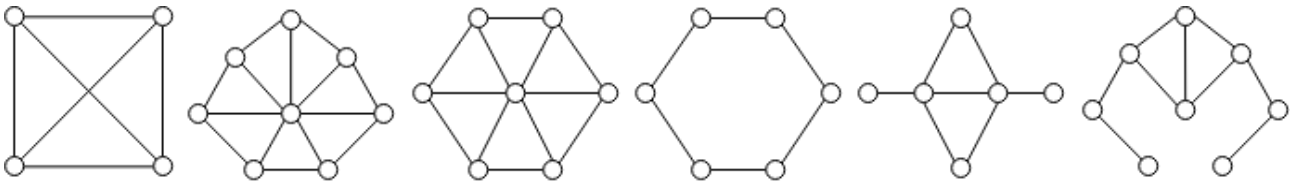
| Your amortized analysis | Mark(s) |
|---|---|
| is wrong | 0 |
| is blank | 0.5 |
| ~~only works for deletion~~ | ~~4~~ |
| only works for insertion | 11 |
| works for both insertion and deletion | 15 |

**Tips:** Focus on only insertion (11 marks) if considering deletion complicates your amortized analysis.

Box B.2.1. Incorrect Attempt

Box B.2.2. Correct Algorithm

Box B.3.1. Manual Test Cases



Box B.3.2. Prove `3-Clique-Cover` is in NP

Box B.3.3. Prove `3-Clique-Cover` is NP-complete

Box B.3.4.1. Solve Special Case $diam(G) = 1$

Box B.3.4.2. Solve Special Case $diam(G) \geq 6$

Box B.4. Computing Diameter of a Directed Weighted Graph

**Grading scheme:** Your answer will be graded using the 7-categories special marking scheme below.

| Time Complexity | Works-on | Mark(s) |
|---|---|---|
| - | wrong answer (only for **unweighted** graphs, or other issue(s)) | 0 |
| - | blank | 0.5 |
| $\omega(\|V\|^{2.5})$ and $O(\|V\|^4)$ | all general **non-negative** weighted graphs | 6 |
| $O(\|V\|^{2.5})$ | all general **non-negative** weighted graphs | 7 |
| $\omega(\|V\|^3)$ and $O(\|V\|^4)$ | all general weighted graphs | 8 |
| $\omega(\|V\|^{2.5})$ and $O(\|V\|^3)$ | all general weighted graphs | 9 |
| $O(\|V\|^{2.5})$ | all general weighted graphs | 10 |

– END OF PAPER; All the Best –