1 Introduction and Objective

The purpose of this tutorial is to do one last review of the basic concepts taught in CS1020 which are the essential foundation to CS2010. Please try out all the questions in the tutorial, and if you encounter difficulties in answering them, don’t hesitate to clear up all doubts and questions you might have, with the tutor (front-line) or lecturer (if the tutor gives up).

At the start of this first tutorial session of CS2010, the tutor (Jonathan, Nathan, Beier, or Zeng Zhong) will introduce himself and similarly you will have to briefly introduce yourself. Tutor reserves 3% of your course grade for tutorial participation point.

- 0% if you only attend $\leq 5$ out of 10+1 tutorial sessions (we have Deepavali public holiday on Tuesday of Week 13),
- 1% if you are very silent throughout all $>5$ tutorial sessions that you attend,
- 3% for top three most active students in each tutorial group, and
- 2% for the rest.

We will start this tutorial by reviewing some of CS1020 topics and then jumps into the first topic of CS2010: Priority Queue (PQ).
2 Tutorial 01 Questions

Sorting

Q1. Which sorting algorithm is the best to sort the following sequence (stored in an array): {50, 9, 8, 7, 6, 5, 4, 3, 2, 1} in ascending order? Explain your answer!

1. Insertion Sort
   (http://visualgo.net/sorting.html?mode=Insertion&create=50,9,8,7,6,5,4,3,2,1)
2. Quick Sort
   (http://visualgo.net/sorting.html?mode=Quick&create=50,9,8,7,6,5,4,3,2,1)
   Note: Quick Sort as initially taught in CS1020: pivot is always the first element
3. Bubble Sort
   (http://visualgo.net/sorting.html?mode=Bubble&create=50,9,8,7,6,5,4,3,2,1)
4. Selection Sort
   (http://visualgo.net/sorting.html?mode=Selection&create=50,9,8,7,6,5,4,3,2,1)
5. Merge Sort
   (http://visualgo.net/sorting.html?mode=Merge&create=50,9,8,7,6,5,4,3,2,1)

Ans: The safest answer given these options is 5. Merge Sort.
Note: \(O(n)\) Counting sort is possible but it is not listed as one of the option.

Insertion, Selection, and Bubble Sort will all take \(O(n^2)\) time since each number is basically in the wrong position.

Quick Sort (as initially taught in CS1020) will also need \(O(n^2)\) since the one of the subarray about the pivot is basically empty thus we perform \(n+(n-1)+(n-2)+...+1 = O(n^2)\) operations. However, if we always randomize the pivot used in partitioning, randomized Quick Sort can be made to run in expected \(O(n \log n)\) In CS3230, you will spend time to do proper analysis of randomized Quick Sort.

Merge Sort will perform \(O(n \log n)\) regardless of whether the starting order of the sequence. This is the safest theoretical answer although may not be the fastest in practice.

After reviewing CS1020 sorting algorithms, please get ready to compare them with Heap Sort and later (balanced) BST Sort.

Order of Growth

Q2.a). What is the bound of the following function? \(F(n) = n + \frac{1}{2}n + \frac{1}{3}n + \frac{1}{4}n + ... + 1\)

1. \(O(2^n)\)
2. $O(n^2)$
3. $O(n \log n)$
4. $O(n)$
5. $O(\log^2 n)$
6. $O(\log n)$

Q2.b). What about $G(n) = n + \frac{1}{2}n + \frac{1}{4}n + \frac{1}{8}n + ... + 1$

Ans: 3. $O(n \log n)$ for $F(n)$
but 4. $O(n)$ for $G(n)$.

$F(n) = n + \frac{1}{2}n + \frac{1}{4}n + ... + 1$

$= n * (\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n})$ $\gg\gg$ there are $n$ terms here

$= n * \sum_{x=1}^{n} \frac{1}{x}$ $\gg\gg$ this is a divergent Harmonic series (we can memorize this)

$= n * O(\ln n)$
$= O(n \log n)$

*Proof* that $\sum_{x=1}^{n} \frac{1}{x} = O(\log n)$

$\sum_{x=1}^{n} \frac{1}{x}$ is the sum of the area of the rectangles which is bounded by the area under the curve $y = \frac{1}{x}$ as shown in fig. 1 (stolen from Wikipedia).

![Figure 1: Area under the curve](image)

Area under the curve $y = \frac{1}{x}$ can be expressed as the integral $\int_{1}^{n} \frac{1}{x} dx$, and $\int_{1}^{n} \frac{1}{x} dx = \ln n$. Thus $\sum_{x=1}^{n} \frac{1}{x} < c * \ln n$ for some constant $c$ (we can choose a $c$ such that the curve is shifted upwards to completely cover the rectangles).

Thus $\sum_{x=1}^{n} \frac{1}{x} = O(\ln n) = O(\log n)$


$G(n) = n + \frac{1}{2}n + \frac{1}{4}n + ... + 1$

$= n * (\frac{1}{1} + \frac{1}{2} + \frac{1}{4} + ... + \frac{1}{\log_2 n})$ $\gg\gg$ there are only $\log_2 n$ terms here, contrast this with Q2.a

$= n * 2$ $\gg\gg$ this is a geometric series that converges to 2

$= O(n)$

Hash Table

Q3. Which of the following is the best hash function?

1. Index = (currentTimeMillis() * (Key.charAt(0) - 'A')) % N;
2. Index = (Key.charAt(0) - 'A') % N;
3. Index = Key.hashCode() % N;

where

- Key is a Java string
- currentTimeMillis() is a function in Java.lang.System that returns the current time in milliseconds
- N is the hash table size, usually a prime number

Ans: The best answer given these three options is 3. \textbf{Index} = \textbf{Key}.hashCode() \% \textbf{N}.

1. This is a non-deterministic hash. Impossible to be used as hash function.

2. The hash values might not be uniformly distributed since it only uses the 1st letter and there are only 26 possible alphabets. Extreme example: What if Keys are NUS matric numbers that all start with character ‘A’?

3. \textbf{hashCode()} is a base31 computation: \texttt{Key[0] \times 31^{(N-1)} + Key[1] \times 31^{(N-2)} + ... + Key[N-1]}. It is deterministic, that is, given the same string it will always give the same hashcode and it has a good distribution since it uses the entire string (each character and its position) to compute the hashcode. And if you are wondering why base 31 is used, the reasons are:
   1. It has to be > 26, the number of alphabets in English
   2. It has to be prime
   Btw, if you use Java String \texttt{hashCode()}, do not set Hash Table size \texttt{N} = 31... Do you know why?
   Btw again, do you realize that Java String \texttt{hashCode()} CAN generate negative hash value? How to deal with that?

Finding \textit{k}-th Smallest Element (Selection Algorithm)

Q4. In the first lecture you have been \textbf{briefly exposed} to the concept of quicksort’s partitioning algorithm combined with binary search-like algorithm on an unsorted array to find the \textit{k}-th smallest element in the array. In this tutorial, we will spend some time discussing the details. Before attending this tutorial, please investigate this algorithm from the Internet: [http://en.wikipedia.org/wiki/Quickselect](http://en.wikipedia.org/wiki/Quickselect).
Ideas (if you have CP3, read Section 9.29, page 380-382):

1. We need a partition function \((\text{Randomized\_Partition})\) to partition a given range \([\text{start}, \text{end}]\) of the array about the given pivot, where all values \(\leq \text{array}[\text{pivot}]\) are placed before the pivot and all values \(> \text{array}[\text{pivot}]\) are placed after pivot. The final position of the pivot, \(\text{PivotIndex}\), is returned.

2. Next employ a standard binary-search-like method (a Divide and Conquer paradigm) to search the array. We start by using a random pivot element and calling the partition function. The position returned by the partition function indicates the final correct rank of the pivot element. If the rank is \(k-1\) (assuming that \(k\) is 1-based), we return the pivot element. If the rank is \(> k-1\), pass the range \([\text{left}, \text{PivotIndex}-1]\) into the algorithm and repeat. If the rank is \(< k-1\), pass the range \([\text{PivotIndex}+1, \text{right}]\) into the algorithm and repeat.

3. The time complexity is \(\text{expected } O(n)\). The detailed proof is probably beyond CS2010 (your CS3230 lecturer may re-use this example).

To be discussed by tutors:
1. What is the best case for this Partition algorithm (regardless it is randomized or not)?
2. As with Quick Sort, what happen to this QuickSelect algorithm if we do NOT randomize the pivot of partition?

Implementation (FYI):

\begin{verbatim}
Algorithm 1 Select(A, l, r, k)
  if l == r then
    return A[l]
  end if
  PivotIndex = Randomized_Partition(A, l, r)  ◢ The pivot is randomly selected between l and r
  if PivotIndex == k-1 then
    ◢ Assume that k is 1-based
    return A[PivotIndex]
  else if PivotIndex > k-1 then
    return Select(A, l, PivotIndex-1, k)  ◢ We have to search on the left side
  else
    return Select(A, PivotIndex+1, r, k)  ◢ We have to search on the right side
  end if
\end{verbatim}

Basic Operations of Binary Heap

Q5. We will end this tutorial 01 with a review of the first topic of CS2010. All 5 basic operations of Binary Heap can be reviewed in \url{http://visualgo.net/heap.html}. During the tutorial session, the tutor will randomize the binary heap structure, ask student to Insert random integers, perform ExtractMax operations (or the first few steps of heapsort), and/or the \(O(n \log n)\) or the \(O(n)\) build heap operation from a random array.
This part is open ended, up to the tutor.
Recommendation: Discuss stuffs that are purposely not implemented in VisuAlgo yet but asked in PS1, the IncreaseKey (or generally: UpdateKey) operation and/or the Extract/Delete Binary Heap element at arbitrary position (not just the max/root element).