CS1102s Data Structures and Algorithms

13/3/2009

Examination Questions Midterm 2

This examination question booklet has 7 pages, including this cover page, and contains 15 questions.

You have 35 minutes to complete the exam. Use a 2B pencil to fill up the provided MCQ form. Leave Section A blank. Fill up Sections B and C.

Trees

Question 1: Which one of the following rates of growth is correct for **the number of leaves** $L(N)$ in a binary tree with N nodes?

1 $\boxed{\text{A}}$ $L(N) = \Theta(N)$

1 $\boxed{\text{B}}$ $L(N) = \Theta(\log N)$

1 $\boxed{\text{C}}$ $L(N) = \Theta(N \log N)$

1 $\boxed{\text{D}}$ $L(N) = \Theta(N^2)$

1 $\boxed{\text{E}}$ none of the above

Answer 1:

1 $\boxed{\text{A}}$ The number of leaves in a binary tree with N nodes is $\lceil N/2 \rceil = \Theta(N)$.

Question 2: Which one of the following statements about the height of an arbitrary binary tree is correct?

2 $\boxed{\text{A}}$ The height of a binary tree with N nodes could be $\log N$, but not more.

2 $\boxed{\text{B}}$ The height of a binary tree with N nodes could be $\log N - 1$, but not more.

2 $\boxed{\text{C}}$ The height of a binary tree with N nodes could be $\log(N - 1)$, but not more.

2 $\boxed{\text{D}}$ The height of a binary tree with N nodes could be N, but not more.

2 $\boxed{\text{E}}$ The height of a binary tree with N nodes could be $N - 1$, but not more.

Answer 2:

2 $\boxed{\text{E}}$ In the worst case, a binary tree could have the shape of a linked list, see Figure 4.12, Page 108. In that case, the height is the number of edges, which is $N - 1$.

Question 3: Consider AVL trees whose nodes are integers, and heaps of integers, i.e. complete binary trees whose nodes are integers and meet the heap-order property. Which one of the following statements is correct?

3 $\boxed{A}$ Every non-empty AVL tree of integers is a heap of integers, but not vice versa.

3 $\boxed{B}$ Every non-empty heap of integers is an AVL tree of integers, but not vice versa.

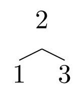
3 $\boxed{C}$ Every non-empty AVL tree of integers is a heap of integers, and vice versa.

3 $\boxed{D}$ none of the above

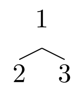
Answer 3:

3 $\boxed{D}$

- Counter example for A: The tree



  is a non-empty AVL tree of integers, but not a heap because $2 > 1$.
- Counter example for B: The tree



  is a non-empty heap, but not an AVL tree because $2 > 1$.
- Both of the above are counter examples for C.

Thus, "none of the above" is correct.

Question 4: Insertion of a new element into an AVL tree may result in a violation of the balance condition of AVL trees. In such a case a (single or double) rotation restores the balance condition. According to the lectures, a rotation is performed on the first node on the path from the inserted element to the root that is in violation of the AVL balance condition. What would happen if we would instead choose the first node on the path from the root to the inserted element that is in violation of the AVL balance condition?

4 $\boxed{A}$ After the rotation, at least one node will be in violation of the balance condition, and thus another rotation will always be necessary.

4 $\boxed{B}$ The entire AVL tree will be balanced after the rotation, but more nodes may be changed by the rotation, and thus the rotation may require more time.

4 $\boxed{C}$ The entire AVL tree will be balanced after the rotation, and the same number of nodes will be changed.

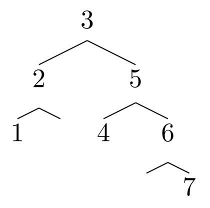
4 $\boxed{D}$ The entire AVL tree will be balanced after the rotation, and the same number or fewer nodes may be changed by the rotation.

4 $\boxed{E}$ The entire AVL tree may or may not be balanced after the rotation.

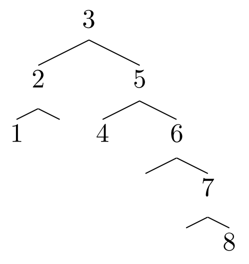
Answer 4:

4 $\boxed{E}$ If there is only one node that violates the AVL balance condition, both versions pick the same node, and we know that after the rotation the balance condition will hold. Thus if we choose the highest node, the tree may be balanced after the rotation.

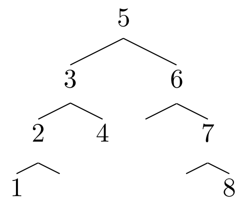
However, the following example shows that the tree may not be balanced after the rotation. Consider insertion of the number 8 into the following tree.



The result of the insertion is the following tree.



The highest node that violates the balance condition is the root, and rotation at the root leads to the following tree.



This tree still has a node that does not meet the balance condition, namely 6.

Hash Tables

Question 5: In separate chaining hash tables, array entries are linked lists. Consider a variant, where second-level hash tables are used instead of linked lists. In such a hash table, a first hashing operation returns a hash table on which a second hashing operation is performed in order to retrieve the required answer. You may assume that the second-level hash tables use linear probing. Which one of the following statements about such two-level hash tables is correct?

5 $\boxed{A}$ The hash function used in the second level should be the same as the hash function used in the second level.

5 $\boxed{B}$ The hash function used in the second level should be different from the hash function used in the first level.

5 $\boxed{C}$ The hash functions may be the same or different, as long as they both have few collisions.

5 $\boxed{D}$ Two-level hash tables are not practically possible, because an array cannot contain arrays as elements.

Answer 5:

5 $\boxed{B}$ If the hash function in the second level is the same as the hash function in the first level, then every insertion after the first one into a second-level hash table will lead to a collision; the second-level hash tables would deteriorate to linked lists.

Question 6: In hash tables without linked lists, a collision resolution function f is used as follows:

$$h_i(x) = (hash(x) + f(i)) \mod \mathit{TableSize}$$

Hash table insertion tries the positions $h_0(x), h_1(x), \ldots$ until an empty array cell is found. Consider the following collision resolution function:

$$f(i) = 2i$$

Which one of the following statements is true, when considering hash tables of a size that is a prime number, compared to the more common linear probing function

$$f(i) = i$$

6 $\boxed{\text{A}}$ The function $f(i) = 2i$ results on average in a reduction of clustering, when compared to $f(i) = i$.

6 $\boxed{\text{B}}$ The function $f(i) = 2i$ results on average in an increase of clustering, when compared to $f(i) = i$.

6 $\boxed{\text{C}}$ The function $f(i) = 2i$ represents neither a reduction nor an increase of clustering, when compared to $f(i) = i$; on average the amount of clustering is the same.

6 $\boxed{\text{D}}$ This conflict resolution function is problematic, because insertion may fail although there are free cells in the array.

Answer 6:

6 $\boxed{\text{C}}$ The clustering is the same; a cluster in this case is a chain of elements at index k, $k + 2$, $k + 4$ etc.

Alternative D is not correct. Since the array size is prime, it will be odd (if it is not 2), and therefore hashing will iterate through all array indices in the worst case. The case where the array size is 2 can be ignored; hash tables of size 2 can be easily avoided.

Question 7: Consider an application where a very large hash table is required, and memory chips are purchased to make sure the hash table fits into main memory. The purchase is such that the expected load factor λ will be between 0.6 and 0.7. Which one of the following statements is true, when considering hash tables of a size that is a prime number?

7 $\boxed{\text{A}}$ Quadratic probing is to be preferred over linear probing in this situation, because primary clustering in linear probing is worse than secondary clustering in quadratic probing.

7 $\boxed{\text{B}}$ Linear probing is to be preferred over quadratic probing, because secondary clustering in quadratic probing is worse than primary clustering in linear probing.

7 $\boxed{\text{C}}$ Linear probing is to be preferred over quadratic probing, because in quadratic probing, insertion may fail even if $\lambda \leq 0.7$.

7 $\boxed{\text{D}}$ Quadratic probing is to be preferred over linear probing, because in linear probing, insertion may fail even if $\lambda \leq 0.7$.

Answer 7:

7 $\boxed{\text{C}}$ Quadratic probing cannot guarantee that insertion is successful, in case $\lambda > 0.5$. Therefore, linear probing (which works until λ gets close to 1) must be used in this situation.

Binary Heaps

Question 8: Consider binary heaps as described in the lectures that support the efficient deletion of the smallest element. Which one of the following statements on binary heaps is **not** correct?

8 $\boxed{\text{A}}$ A binary heap has at most one node with exactly one child.

8 $\boxed{\text{B}}$ In a node of a binary heap with two children, the value of the right child is always smaller than the value of the left child.

8 $\boxed{\text{C}}$ If a node of a binary heap has a right child, it also has a left child.

8 $\boxed{\text{D}}$ The root of a non-empty binary heap always contains the smallest element.

8 $\boxed{\text{E}}$ One of the leaves of a non-empty binary heap always contains the largest element.

Answer 8:

8 $\boxed{\text{B}}$ The heap property does not stipulate any order between left and right child of a node. All other statements are correct.

Question 9: In order to build a heap from a given array of elements, the class BinaryHeap copies the elements into an array starting at index 1, and then calls the following function buildHeap().

```
private void buildHeap() {
    for (int i = currentSize / 2; i > 0; i--)
        percolateDown(i);
}
```

Note that the index i counts backwards. Consider the following alternative implementation, where the index i counts forwards.

```
private void buildHeap() {
    for (int i = 1; i <= currentSize / 2; i++)
        percolateDown(i);
}
```

Which one of the following statements is correct?

9 $\boxed{\text{A}}$ This alternative implementation is incorrect; it may or may not result in a heap.

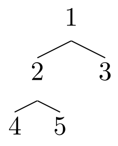
9 $\boxed{\text{B}}$ This alternative implementation is correct and as efficient as the original version.

9 $\boxed{\text{C}}$ This alternative implementation is correct, but not as efficient as the original version.

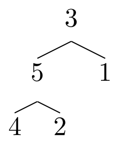
9 $\boxed{\text{D}}$ This alternative implementation is correct, and in some situations more efficient and in other situations less efficient than the original version.

Answer 9:

9 $\boxed{\text{A}}$ Consider for example the following binary tree.



Using the alternative buildHeap function, we will get



which is not a heap.

Question 10: Which one of the following rates of growth is correct for the worst-case runtime $T(N)$ of the insert operation on binary heaps of size N?

10 $\boxed{\text{A}}$ $T(N) = \Theta(N)$

10 $\boxed{\text{B}}$ $T(N) = \Theta(N^2)$

10 $\boxed{\text{C}}$ $T(N) = \Theta(\log N)$

10 $\boxed{\text{D}}$ $T(N) = \Theta(N \log N)$

10 $\boxed{\text{E}}$ none of the above

Answer 10:

10 $\boxed{\text{C}}$ In the worst case, the function percolateUp will reach the root of the tree. Thus $\log N$ steps are required.

Question 11: Which one of the following rates of growth is correct for the worst-case runtime $T(N)$ of the deleteMin operation on binary heaps of size N?

11 $\boxed{\text{A}}$ none of the below

11 $\boxed{\text{B}}$ $\Theta(N^2)$

11 $\boxed{\text{C}}$ $\Theta(N)$

11 $\boxed{\text{D}}$ $\Theta(1)$

11 $\boxed{\text{E}}$ $\Theta(\log N)$

Answer 11:

11 $\boxed{\text{D}}$ The operation deleteMin moves the last element of the heap into the root, and calls percolateDown. This operation may reach in the worst case the level just above a leaf. Thus $T(N) = \Theta(\log N)$.

Sorting

Question 12: What is the number of element comparisons (calls of compareTo(...)) performed by insertion sort when applied to arrays of size N that are already correctly sorted?

12 $\boxed{\text{A}}$ $N - 1$

12 $\boxed{\text{B}}$ $N - 2$

12 $\boxed{\text{C}}$ N

12 $\boxed{\text{D}}$ $N^2 - 1$

12 $\boxed{\text{E}}$ N^2

Answer 12:

12 $\boxed{\text{A}}$ The comparison inn the line

```
for (j = p; j > 0 && tmp.compareTo(a[j-1]) < 0; j--)
```

will always fail, and thus comparison will be performed only as often as the outer loop runs, which is $N - 1$ times.

Question 13: An inversion in an array a is a pair of array indices (i, j) such that $i < j$ but $a[i] > a[j]$. What is the maximal number of inversions that can be eliminated by the following program fragment?

```
AnyType tmp = a[5];
a[5] = a[10];
a[10] = tmp;
```

13 $\boxed{\text{A}}$ 8

13 $\boxed{\text{B}}$ 9

13 $\boxed{\text{C}}$ 10

13 $\boxed{\text{D}}$ 13

13 $\boxed{\text{E}}$ 20

Answer 13:

13 $\boxed{\text{B}}$ In the best case, $a[5]$ is larger than $a[6]$ through $a[10]$, and $a[10]$ is smaller than $a[5]$ through $a[9]$. In this case, the following nine inversions are eliminated: $(5,6)$, $(5,7)$, $(5,8)$, $(5,9)$, $(5,10)$, $(6,10)$, $(7,10)$, $(8,10)$, and $(9,10)$.

Question 14: Which one of the following rates of growth is correct for the runtime $T(N)$ of Shellsort using the increment sequence $\{1\}$ (only one *phase*), when applied to an arbitrary array of size N?

14 $\boxed{\text{A}}$ $T(N) = \Theta(1)$

14 $\boxed{\text{B}}$ $T(N) = \Theta(N)$

14 $\boxed{\text{C}}$ $T(N) = \Theta(N^{3/2})$

14 $\boxed{\text{D}}$ $T(N) = \Theta(N^2)$

14 $\boxed{\text{E}}$ none of the above

Answer 14:

14 $\boxed{\text{D}}$ If there is only one phase, Shellsort is identical to insertion sort, which has a runtime $T(N) = \Theta(N^2)$.

Question 15: Which one of the following rates of growth is correct for the runtime $T(N)$ of heapsort, when applied to arrays of size N that are sorted in reverse order?

15 $\boxed{\text{A}}$ none of the below

15 $\boxed{\text{B}}$ $T(N) = \Theta(N)$

15 $\boxed{\text{C}}$ $T(N) = \Theta(N \log\log N)$

15 $\boxed{\text{D}}$ $T(N) = \Theta(N \log N)$

15 $\boxed{\text{E}}$ $T(N) = \Theta(N^2)$

Answer 15:

15 $\boxed{\text{D}}$ The buildHeap phase takes $\Theta(N)$ time (in this case percolateDown always stops immediately).

Each call of deleteMax replaces the largest element by the smallest element, which has to percolateDown to a leaf. Thus, we have $T(N) = \Theta(N \log N)$.