CSc 345 Midterm 2

Name	
UA ID Number	
UA email	

General Instructions.

- Put your name and email in the space provided.
- For all questions, read the instructions carefully and write legibly. Illegible answers will not receive credit.
- Do not tear any pages from the exam.
- You may write on any portion of the exam, but only answers in the indicated spaces or in the designated "Extra Space" areas will be graded.
- Assume that all logarithms are base-2 unless specifically indicated otherwise.
- Please read all multiple choice answers carefully.
- Note that when asked for the best-case runtime, you should give the tightest lower bound, and when asked for the worst-case runtime, you should give the tightest upper bound.
- If you need extra space, you can use the last page of the exam, but make sure you make a note in the question that you used the extra space. Also, make sure the work is labeled with the appropriate Section and Question number.
- You are not allowed to use a calculator for this exam.

Some formulas that might be useful:

Summations:

$$\sum_{k=1}^{N} k = \frac{N(N+1)}{2}$$

$$\sum_{k=0}^{N} r^k = \frac{r^{N+1}-1}{r-1},\ r \neq 1$$

The Master Theorem: Given a function of the form $f(N) = af(N/b) + cN^d$, $f(N)$ is

- $O(N^d)$ if $a < b^d$
- $O(N^d logN)$ if $a = b^d$
- $O(N^{log_b a})$ if $a > b^d$

Other: $log(N!)\ is\ O(NlogN)$

Part 1. Short Answer. Answer each question in the space provided.

1. (15 points) Answer the questions about the pseudocode below. At the right, you are given the printout of an input array as it passes through the algorithm. (Hint: This is either Quicksort or Mergesort.)

```
1  Input: A, an array of N integers
2  Output: ???
3  foo(A, 0, N-1)
4
5  proc foo(Array A, int i, int j):
6  print A
7  if(j <= i)
8     return
9  int k = i
9  int m = k
10 int num = A[i]
11 while(m <= j):
12     if A[m] < num
13        if m != k
14           swap A[k] and A[m]
15        end if
16        k++
17     end if
18    m++
19 end while
20 foo(A, i, k-1)
21 if(A[k] == num)
22    k++
23 foo(A, k, j)
```

```
Input Array:
[5, 1, 2, 7, 8, 9, 0, 3, 4, 6]

What is printed by line 6:
[5, 1, 2, 7, 8, 9, 0, 3, 4, 6]
[1, 2, 0, 3, 4, 9, 5, 7, 8, 6]
[0, 2, 1, 3, 4, 9, 5, 7, 8, 6]
[0, 2, 1, 3, 4, 9, 5, 7, 8, 6]
[0, 1, 2, 3, 4, 9, 5, 7, 8, 6]
[0, 1, 2, 3, 4, 9, 5, 7, 8, 6]
[0, 1, 2, 3, 4, 9, 5, 7, 8, 6]
[0, 1, 2, 3, 4, 9, 5, 7, 8, 6]
[0, 1, 2, 3, 4, 9, 5, 7, 8, 6]
[0, 1, 2, 3, 4, 5, 7, 8, 6, 9]
[0, 1, 2, 3, 4, 5, 7, 8, 6, 9]
[0, 1, 2, 3, 4, 5, 7, 8, 6, 9]
[0, 1, 2, 3, 4, 5, 6, 8, 7, 9]
[0, 1, 2, 3, 4, 5, 6, 8, 7, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

(a) Which sorting algorithm is this?
(b) Explain the non-trivial* best-case runtime of this algorithm. Your explanation should include: (i) A recurrence relation with an explanation of each part and an explanation of how it relates to the pseudocode (ii) a description of the type of input that would result in this runtime with an explanation of why it would result in that runtime (iii) the runtime in big-Oh notation with a justification based on the recurrence relation.
(c) Explain the worst-case runtime of this algorithm. Your explanation should include: (i) A recurrence relation with an explanation of each part and an explanation of how it relates to the pseudocode (ii) a description of the type of input that would result in this runtime with an explanation of why it would result in that runtime (iii) the runtime in big-Oh notation with a justification based on the recurrence relation.
(d) Explain how the worst case is generally avoided in practice. You do not have to justify this mathematically.
*I am not looking for the runtime when all the elements are equal.

(a) Quicksort

(b) In the best case, the pivot that is used is always the median, which will divide the array into halves. In the psuedocode, the final *k* represents where this division happens, and the two recursive calls are done on the two parts–what is to the left of *k* and what is to the right of *k*. If the *k* evenly divides the two halves, then the input sizes of the two recursive calls are N/2. In addition. The pivoting process takes O(N) time and the the base case is O(1), which is when the subarray contains 1 element or less. That means that the recurrence relation would be T(N) = 2T(N/2) + N; T(1) = 1. Using the Master Theorem, a = 2, b = 2, and d = 1 and $2 = 2^1$, so the runtime is O(NlogN).

(c) In the worst case, the pivot is always the min or the max, so if the array is already sorted in either direction, we will get the worst case. In this case, the pivot divides the array into arrays of size 1 and N-1, so one recursive call will just be T(1) = 1 (the base case) and the other will be T(N-1). We still do O(N) work for the pivoting and the base case is T(1) = 1, so the recurrence relation is T(N) = T(N-1) + N; T(1) = 1. To solve this, we can expand it out and use a summation.
T(N) = T(N-1) + N = T(N-2) + N-1 + N = T(N-3) + N-2 + N-1 + N = … = 1 + 2 + … + N-2 + N-1 + N, which is $O(N^2)$.

(d) The worst case is generally avoided through randomization. If the pivot is random, then we can expect about half the pivots to be good pivots and the expected runtime is O(NlogN).

2. (8 points) Recall the following definitions for the number of external nodes *e(T)* and the total number of nodes *n(T)* of a full binary tree.

Definition 1. Let $n(T)$ be the total number of nodes in a full binary tree T.

Basis Step. $n(T_0) = 1$ where T_0 is a full binary tree that is just the root.

Inductive Step. Given that T_1 and T_2 are full binary trees, we can combine the two into a new full binary tree called T_3 by making each tree a child of a new root. Then $n(T_3) = n(T_1) + n(T_2) + 1.$

Definition 2. Let $e(T)$ be the number of external nodes in a full binary tree T.

Basis Step. $e(T_0) = 1$ where T_0 is a full binary tree that is just the root.

Inductive Step. Given that T_1 and T_2 are full binary trees, we can combine the two into a new full binary tree called T_3 by making each tree a child of a new root. Then $e(T_3) = e(T_1) + e(T_2)$.

Prove by structural induction that $n(T) = 2e(T) - 1$ for all full binary trees.

Basis Step. Let T_0 be a full binary tree with just the root. Then $e(T_0) = 1$ and $n(T_0) = 1$ and $1 = 2(1) - 1.$

Inductive Step. Let T_1 and T_2 be full binary trees. Assume as the IH that $n(T_1) = 2e(T_1) - 1$ and $n(T_2) = 2e(T_2) - 1$. We can form a new full binary tree T_3 by making T_1 and T_2 each the child of a new root. Then

$n(T_3) = n(T_1) + n(T_2) + 1$

$= 2e(T_1) - 1 + 2e(T_2) - 1 + 1$ by the IH

$= 2(e(T_1) + e(T_2)) - 1$

$= 2e(T_3) - 1$

3. (7 points) Show the following array as it passes through the Heapsort algorithm. You should show the array contents after each *swap*. Note: I've given you exactly enough space to do this, so if you find yourself not filling the whole grid or needing more rows, you are probably doing something wrong. I've provided indexes for your reference. These start at index 1, so for a node at index *k*, the children will be at *2k* and *2k+1.*

Input Array: `[7, 1, 0, 9, 2]`

1	2	3	4	5
7	1	0	9	2
7	9	0	1	2
9	7	0	1	2
2	7	0	1	9
7	2	0	1	9
1	2	0	7	9
2	1	0	7	9
0	1	2	7	9
1	0	2	7	9
0	1	2	7	9

[Extra Space]

Part 3. Multiple Choice. (4 points each) For each question, *circle the letter of the best response.* Each question has one best answer. *Do not circle multiple answers.* You are not required to show your work. However, if your answer is wrong and you do show your work, your work *may* earn you some partial credit.

Questions 1-3 refer to the proof below.
Definitions for height *h(T)* and number of internal nodes *i(T)* of a full binary tree.
Let T_0 be a full binary tree with just one node. Let T_1 and T_2 be full binary trees and let T_3 be the full binary tree made by making T_1 and T_2 the children of a new root. Then:
$h(T_0) = 0$ and $h(T_3) = max(h(T_1), h(T_2)) + 1$
$i(T_0) = 0$ and $i(T_3) = i(T_1) + i(T_2) + 1$

Conjecture: For any full binary tree T, $h(T) \leq i(T)$.
Proof.
1 Let T_0 be a full binary tree with just the root. $h(T_0) = 0 \leq 0 = i(T_0)$.
2 Let T_1 and T_2 be full binary trees and
3 let T_3 be T_1 and T_2 joined by a new root.
4 $h(T_3) = max(h(T_1), h(T_2)) + 1$
5 $\leq h(T_1) + h(T_2) + 1$
6 $\leq i(T_1) + i(T_2) + 1$
7 $= i(T_3)$

1. In which line(s) is the base case proven?
[✓ A. Line 1] B. Lines 2-3 C. Line 4 D. Line 5 E. None–the base case is not proven.

2. In which line(s) is the inductive hypothesis applied?
A. Line 1 B. Line 4 C. Line 5 [✓ D. Line 6] E. Line 7

3. Is this proof logically valid and correct? Keep in mind that this question is not asking whether the proof includes all the aspects that may help with clarity. It is asking if it correctly shows the reasoning required for a valid inductive proof.
[✓ A. Yes] B. No

4. What is the in-order traversal of the max binary heap that results from the following

operations? Note that the heap starts as empty.
`insert 8, insert 1, insert 9, insert 2, insert 10, delMax, insert 4`

A. `1-4-2-9-8`
B. `1-2-4-8-9`
C. `9-8-4-2-1`
D. `1-2-4-9-8`
E. None of the above.

5. Which of the following statement(s) is/are true about min binary heaps?
A. Given a min binary heap of size *N* and an element *x* to search for, you can retrieve *x* in worst case O(logN) time.
B. Given a min binary heap of size *N*, you can retrieve the minimum value in worst case O(logN) time.
C. Given a min binary heap of size *N*, you can retrieve the maximum value in worst case O(logN) time.
D. Given a min binary heap of size *N*, you can insert a new item into the heap in worst case O(logN) time.
E. Both A and B.
F. Both B and D.
G. Both C and D.
H. All of the above.
I. None of the above.

6. Consider the pseudocode below.

```
S := an empty Stack
Q := an empty Queue
for i from 1 to N:
   Q.enqueue(i)
end for
while(!Q.isEmpty())
   S.push(Q.dequeue())
end while
while(!S.isEmpty())
   print S.pop()
end while
```

In which order will the numbers 1 to N be printed?
A. Increasing order from 1 to N.
B. Decreasing order from N to 1.
C. Random order.
D. It's impossible to know.
E. None of the above.

7. Consider a Queue implemented as a wrap-around array as discussed in class. If the size of the underlying array is 6, at which array index is the 9 after the following operations have been executed? Assume the Queue starts out empty.

```
enqueue 0, enqueue 1, enqueue 2, enqueue 3, dequeue, dequeue, enqueue 4,
enqueue 5, dequeue, dequeue, enqueue 6, enqueue 7, enqueue 8, dequeue,
dequeue, enqueue 9
```

A. 0 B. 1 C. 2 D. 3 E. 4 F. 5

8. The numbers 0 to 9 are pushed onto an empty Stack in order. They are also popped off, but the exact sequence of push and pop calls is unknown. For example, the sequence below is one possible sequence of push/pop calls. Each item is only pushed to the Stack and popped from the Stack exactly once.

```
push 0, push 1, pop, pop, push 2, push 3, pop, push 4, push 5, push 6, push
7, push 8, push 9, pop, pop, pop, pop, pop, pop, pop
```

Which of the following is not a possible order in which the numbers are popped from the Stack?

A. 0 1 2 3 4 5 6 7 8 9
B. 9 8 7 6 5 4 3 2 1 0
C. 1 0 3 2 5 4 7 6 9 8
D. 0 9 1 8 2 7 3 6 4 5
E. None of the above–all the orders are possible.

9. The numbers 0 to 9 are enqueued to an empty Queue in order. They are also dequeued, but the exact sequence of enqueue and dequeue calls is unknown. For example, the sequence below is one possible sequence of enqueue/dequeue calls. Each item is only enqueued to the Queue and dequeued from the Queue exactly once.

```
enqueue 0, enqueue 1, dequeue, dequeue, enqueue 2, enqueue 3, dequeue,
enqueue 4, enqueue 5, enqueue 6, enqueue 7, enqueue 8, enqueue 9, dequeue,
dequeue, dequeue, dequeue, dequeue, dequeue, dequeue
```

Which of the following is a possible order in which the numbers are popped from the Stack?

A. 9 8 7 6 5 4 3 2 1 0
B. 9 0 8 1 7 2 6 3 5 4
C. 0 1 2 3 5 4 6 7 8 9
D. 2 1 0 3 4 5 9 8 7 6
E. None of the above.

Questions 10 - 13 refer to the pseudocode below.

```
S := an empty Stack
for i from 1 to N:
   S.push(i)
end for
```

10. Assume that *S* is implemented as an array with a starting capacity of 1. When an item is pushed to *S* and the array is full, the array is doubled in size. If the array is full after the *N*th *push*, what is the total number of array accesses from the pseudocode?

A. $\frac{3N-1}{2}$ [✓ B. $3N - 2$] C. $N^2 - 1$ D. $\frac{N(N+1)}{2}$ E. Answer not listed.

11. If *S* is implemented as described in #10, what is the runtime of the pseudocode?

A. $O(1)$ B. $O(\log N)$ [✓ C. $O(N)$] D. $O(N^2)$ E. Answer not listed.

12. If *S* is implemented as an array like in #10 except that instead of doubling it increases by 100 when it's full, what is the runtime of the pseudocode?

A. $O(1)$ B. $O(\log N)$ C. $O(N)$ [✓ D. $O(N^2)$] E. Answer not listed.

13. If *S* is implemented as a linked list, what is the runtime of the pseudocode?

A. $O(1)$ B. $O(\log N)$ [✓ C. $O(N)$] D. $O(N^2)$ E. Answer not listed.

14. Consider an unknown data structure called *StructureX* that has an operation called *insert*. If the runtime of the pseudocode below is $O(NlogN)$, what is the amortized cost of the *insert* operation?

```
A := an empty StructureX
for i from 1 to N:
   A.insert(i)
end for
```

A. O(1) [✓ B. O(logN)] C. O(N) D. O(NlogN) E. Answer not listed.

Questions 15-17 refer to the pseudocode below. Keep in mind that this Priority Queue is getting very specific input. You can assume that no resizing is necessary for any of these implementations.

```
Q := an empty max priority queue
for(int i = N; i >= 1; i--)
   Q.insert(i)
end for
while(!Q.isEmpty)
   Q.delMax
end while
```

15. If Q is implemented as an unsorted array, what is the runtime of the pseudocode?
A. O(1) B. O(logN) C. O(N) D. O(NlogN) [✓ E. O(N^2)]

16. If Q is implemented as an array that is sorted in increasing order, what is the runtime of the pseudocode?
A. O(1) B. O(logN) C. O(N) D. O(NlogN) [✓ E. O(N^2)]

17. If Q is implemented as a max binary heap, what is the runtime of the pseudocode?
A. O(1) B. O(logN) C. O(N) [✓ D. O(NlogN)] E. O(N^2)

18. Which sorting algorithm is illustrated below?

```
5 6 8 1 2 3
5 6 1 8 2 3
5 1 6 8 2 3
1 5 6 8 2 3
1 5 6 2 8 3
1 5 2 6 8 3
1 2 5 6 8 3
1 2 5 6 3 8
1 2 5 3 6 8
1 2 3 5 6 8
```

A. Bubble Sort B. Selection Sort [✓ C. Insertion Sort] D. Heapsort E. Mergesort

19. Which sorting algorithm is illustrated below?

```
5 6 8 1 2 3
5 6 1 8 2 3
5 6 1 2 8 3
5 6 1 2 3 8
5 1 6 2 3 8
5 1 2 6 3 8
5 1 2 3 6 8
1 5 2 3 6 8
1 2 5 3 6 8
1 2 3 5 6 8
```

[✓ A. Bubble Sort] B. Selection Sort C. Insertion Sort D. Heapsort E. Mergesort

20. Which sorting algorithm is illustrated below?

```
5 6 8 1 2 3
1 6 8 5 2 3
1 2 8 5 6 3
1 2 3 5 6 8
```

A. Bubble Sort [✓ B. Selection Sort] C. Insertion Sort D. Heapsort E. Mergesort

21. Consider a version of Mergesort that divides each array into three subarrays and does a recursive call on each of the thirds. Which of the following recurrence relations would best represent the runtime of such an algorithm?

A. $T(N) = T(N/3) + 1$
B. $T(N) = 3T(N/3) + 1$
[✓ C. $T(N) = 3T(N/3) + N$]
D. $T(N) = T(N/3) + N$
E. $T(N) = 3T(N/3) + logN$

22. Which of the following statements is/are true about Mergesort and Quicksort?
A. Quicksort has a better worst-case runtime than Mergesort.
B. The expected runtime of both algorithms is O(N).
C. The best-case runtime of both algorithms is O(N).
D. Quicksort uses less space than Mergesort.
E. All of the above.
F. None of the above.

23. Alice is writing a method for sorting strings alphabetically. She proposes the algorithm below. Note that strings are arrays of characters. For simplicity, assume all input strings use only lowercase English characters. Note that ‘$’ is considered to be first alphabetically (so it occurs before ‘a’.)

```
1      Algorithm sortStrings
2      Input: An array of N strings
3      Output: The same set of strings, sorted alphabetically
4
5      scan the array and get the length of the longest string
6      call that value m
7      for each string in the array:
8           if the length of the string is less than m:
9                add ‘$’ to the end until the length of the string is m
10          end if
11     end for
12
13     for i from 0 to m-1:
14          sort all the strings according to the ith character using
15          heapsort
16     end for
17
18     scan the array again, removing all trailing ‘$’ from each string
```

Which of the following changes would make the algorithm work as intended (i.e. it correctly sorts the strings alphabetically)?
A. The “$” should be added to the beginning of the words instead of the end.
B. The loop in lines 13 to 16 should go from m-1 down to 0.
C. Change the algorithm in lines 14-15 to Selection Sort.
D. Change the algorithm in lines 14-15 to Mergesort.
E. A, B, and C.
F. Both B and C.
G. Both B and D.
H. None of the above–the algorithm is already correct.

24. Consider a Radix sort on *N* 64-bit binary integers. Which of the following best describes the runtime?
A. $10N + 64$ B. $64N + 10$ C. $64N + 2$ D. $64N + 128$ E. $32N + 1$

25. Which of the following are necessary characteristics of a good hash function (for use in a hashtable)?
A. It is consistent, meaning that if $k_1 = k_2$, then $h(k_1) = h(k_2)$.
B. It is easy to compute and not too expensive.
C. If $h(k_1) = h(k_2)$, then $k_1 = k_2$.
D. It distributes the keys uniformly across possible hash values.
E. A and B
F. B and C
G. A, B, and D
H. All of the above.

Questions 26-27 refer to the hashtable below.

0	1	2	3	4	5	6	7	8	9	10
		13	**14**	**24**	**3**					

The hash function is just using simple modular hashing.

26. If the hashtable manages collisions with linear probing, at which index will key $k = 4$ end up if it gets inserted into the table next?
A. 0 B. 6 C. 9 D. 10 E. Answer not listed.

27. If the hashtable manages collisions with quadratic probing, at which index will key $k = 4$ end up if it gets inserted into the table next?
A. 5 B. 6 C. 7 D. 8 E. Answer not listed.

28. If a hashtable of size *M* and containing *N* key-value pairs manages collisions with separate chaining, what is the expected runtime of the *put* and *get* operations?
A. $O(N/M)$ B. $O(M/N)$ C. $O(N)$ D. $O(M)$ E. $O(NM)$

29. What does the load factor α refer to in a hashtable that uses linear probing or quadratic probing?
A. The average length of the lists.
B. The fraction of the table that is full.
C. The number of elements in the table.
D. The number of un-deleted elements in the table.
E. None of the above.

30. Which of the following describe(s) a scenario in which a hashtable would be a good option?
A. You are storing data and need to be able to *insert* and *search* for items quickly.
B. You are storing data and need to be able to get the *maximum* or the *minimum* quickly.
C. You are storing data and need to be able to get a sorted list of the items quickly.
D. All of the above.
E. None of the above.

Extra Credit (5 points).

What is the maximum height possible of a binary tree containing N nodes? Explain your reasoning.

N-1. In the worst case, the tree would be completely unbalanced and basically just be a linked list.