COM S 311 SPRING 2021
EXAM 1

Due: March 9 7:59 p.m.

You must turn in a single pdf with your typed answers by 7:59.

Haadi-Mohammad Majeed

Guidelines

- For each problem, if you write the statement "I do not know how to solve this problem" (and nothing else), you will receive 20% credit for that problem. If you do write a solution, then your grade could be anywhere between 0% to 100%. To receive this 20% credit, you must explicitly state that you do not know how to solve the problem.
- You are **not** allowed to discuss the problems with anyone. You are allowed to use the textbook and notes. Do **not** copy solutions from the internet. Your writing should demonstrate that you understand the proofs completely.
- When proofs are required, you should make them both clear and rigorous. Do not handwaive.
- Please submit your assignment on the given Canvas exam.
  - You **must** type your solutions. Please submit a PDF version.
  - Please make sure that the file you submit is not corrupted and that its size is reasonable (e.g., roughly at most 10-11 MB).

    If we cannot open your file, your exam will not be graded.
- Any concerns about grading should be expressed within one week of returning the exam.

Problems

(1) Prove or disprove the following statements **(20 points)**.

  (a) $4\sqrt{n} = O(n)$

  $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \lim_{n\to\infty} \frac{4\sqrt{n}}{n} \to 0$. Therefore O(n) is an upper asymptotic bound for $4\sqrt{n}$ Proven.

  (b) $n = O(4\sqrt{n})$.

  $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \lim_{n\to\infty} \frac{n}{4\sqrt{n}} \to \infty$. Therefore O(n) is an not upper asymptotic bound for n Disproven.

(2) Formally analyze the runtime of the following algorithm. Give the runtime in big oh notation. You must show your work. **(20 points)**

```
Alg1(A)
    Input: Array of integers of length n
    constant number of operations
    for i = n, i ≥ 1, i = i/2 do
        for j = 1, j ≤ n, j = j + 1 do
            constant number of operations
```

Runtimes per line
  c - constant operations
  log(n)+1 - for loop that half's iterations
  n+1 - linear for loop
  c - constant operations

Sums to $log(n((n*c)+1))+1+c$ which can be simplified to $O(log(n^2))$

(3) We are given an array A of integers which is *strictly increasing*, i.e., $A[i] < A[i+1]$. Give a divide-and-conquer algorithm which outputs an index i such that $A[i] = i$, if one exists. If no such index exists, the algorithm outputs null. Formally analyze the runtime of your algorithm, giving a recurrence relation and a big oh bound on the runtime of your algorithm. You **must** use a divide and conquer strategy. You do not have to prove correctness. **(30 points)**

```
int algorithm(int arr[], int high, int low){
        if(low <= high){   //runtime of C
                int mid = (high + low) /2; //runtime of C
                if(mid == arr[mid]){ //runtime of C
                        return mid; //runtime of C
                }
                else if(mid > arr[mid]){ //runtime of C
                        //logrithmic runtime
                        return binSearch(arr, high, mid+1);
                }
                else{
                        //logrithmic runtime
                        return binSearch(arr, mid-1, low);
                }
        }
        return -1;
}
```

$T(n) = \log n + 1 + c$
$T(n) = O(\log n)$

(4) Using the Master Theorem, bound the runtime $T(n)$ of the following recurrence.

$$T(n) = 2T(n/4) + 16\sqrt{n} + 1, \text{ where } T(1) = O(1).$$

You must state which case of the Master Theorem holds, and prove that it does apply. **(20 points)**
Case 2
$a = 2$
$b = 4$
$f(n) = \sqrt{n} \quad log_4(2) = 0.5 \rightarrow n^{log_4(2)} log(n) \leq n^{1/2} log(n)$
$\therefore n^{.5} \leq \omega(n^{.5} log(n))$
$a * F(\frac{n}{b}) \leq C * F(n)$ Making C = 1 we get
$2F(\frac{n}{4}) \leq n^{0.5}$
$2F(\frac{n}{4}) \leq 2 * (\frac{n}{4})^{0.5}$
$2F(\frac{n}{4}) \leq n^{0.5}$

$2F(\frac{n}{4}) \leq C * F(n)$
Therefore case 2 applies and we can establish that $T(n) = \theta(n)$

(5) Recall that a *leaf node* of a heap is a node which does not have any children. An *internal node* is a node which is not a leaf, i.e., a node which has at least one child. Prove that the number of leaves in an n-element max-heap is $\lceil n/2 \rceil$. **(10 points)**

Hint: Remember that every heap has an associated array with n elements, starting with index 1, such that, for every $i \in \{1, \ldots, n\}$,

$$\text{Parent}(i) = \lfloor i/2 \rfloor, \text{ Left}(i) = 2i, \text{ and Right}(i) = 2i + 1.$$

To get started on the problem, consider $2i$ and $2i+1$ when $i > \lfloor n/2 \rfloor$ and when $i \leq \lfloor n/2 \rfloor$.

x = depth of tree
There are $2^{x+1} - 1$ nodes within the heap
level x-1 has $2^x - 1$ nodes within it
level x has $n - 2^x + 1$ nodes, all are leaves here, which mean they all have parents at level x-1, of the $2^{x-1} nodes$ $\lceil \frac{n-2^x+1}{2} \rceil$ are parents and $2^{x-1} - \lceil \frac{n-2^x+1}{2} \rceil$ are leaves