COM S 311 EXAM 1

NATHAN TUCKER (NJTUCKER@IASTATE.EDU)

This assignment represents my own work in accordance with University regulations.

- Nathan Tucker

Problems

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

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

By the limit definition of big O, we can say that g(n) is an upper asymptotic bound if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

$$\lim_{n\to\infty} \frac{4\sqrt{n}}{n} = 0$$

Because we know that $\lim_{n\to\infty} \frac{4\sqrt{n}}{n}$ goes to 0, we can say that O(n) is an upper asymptotic bound for $4\sqrt{n}$, or $4\sqrt{n} = O(n)$.

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

By the limit definition of big O, we can say that g(n) is an upper asymptotic bound if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

$$\lim_{n\to\infty} \frac{n}{4\sqrt{n}} = \infty$$

Because we know that $\lim_{n\to\infty} \frac{n}{4\sqrt{n}}$ goes to ∞, we can say that $4\sqrt{n}$ is not an 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)**

1 *Alg1(A)*
  Input: Array of integers of length n
2 constant number of operations
3 **for** $i = n,\ i \geq 1,\ i = i/2$ **do**
4   **for** $j = 1,\ j \leq n,\ j = j + 1$ **do**
5   constant number of operations

1 *Runtime of each line*
2 c
3 logarithmic, potentially log(n) (+ 1 for the final comparison to exit loop)
4 n + 1
5 c * n

The outermost term is log (n), as we are dividing i by 2 each iteration. The next loop is a simple from 1 to n, incrementing by 1, meaning depending on n items (+ 1 for a check that will exit the loop). Then it's a constant number of operations times n, the outer loop. Combined together, we can say that the total runtime of t

$$log(n * [(n * c) + 1]) + 1 + c$$

As the largest leading term in the above runtime is $log(n^2)$ we can say that the big O of the algorithm is $log(n^2)$

(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)**

```
IndexEqualtoValue(A, start, end)
    Input: Array of integers of length n, the left most index, the right most index
    if end >= start then
        middle = ⌊(low+high)/2⌋
        if middle == A[middle] then
            return middle
        if middle > A[middle] then
            return IndexEqualtoValue(A, middle + 1, end)
        if middle < A[middle] then
            return IndexEqualtoValue(A, start, middle)
    return "null"
```

To formlly analyze the runtime of the algorithm, we can see that we are doing a search in a list. This is not a linear search however, as we first search over half the list, then, if we do not find the desired value, we search over half of that half, or $\frac{1}{4}$. Eventually we will get an array length of $\frac{n}{2^k}$ over k iterations. Additionally, we know that, after k iterations, we will end up with an array size of 1. Therefore we can say that the length of the array is $1 = \frac{n}{2^k}$ or, rearranging, $n = 2^k$. Applying log to both sides yields $log_2 n = k * log_2 2$, or $k = log_2 n$. Making the big O = $O(log(n))$ Because of this too, we can say that the recurrence relation of this search is

$$T(n) = T(\frac{n}{2}) + 1$$

(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)**

$a = 2$, $b = 4$, $f(n) = \sqrt{n}$

$log_4 2 = .5$, which yields a c value of .5

Therefore, case 2 of the Master Theorem applies here, of $f(n) = \Omega(n^{log_b a} * log(n))$

Proof:

We see that $a = 2$, $b = 4$, and $f(n) = 16\sqrt{n}$

$log_4 2 = .5 = \frac{1}{2}$, then

$n^{log_4 2} * log(n) \leq n^{\frac{1}{2}} * log(n)$

$\therefore n^{\frac{1}{2}} \leq \Omega(n^{\frac{1}{2}} * log(n))$

$a * f(\frac{n}{b}) \leq c * f(n)$, let c = 1

$2f(\frac{n}{4}) \leq c * n^{\frac{1}{2}}$

$\leq 2 * \frac{n}{4}^{\frac{1}{2}}$

$\leq n^{\frac{1}{2}}$

$\leq c * f(n)$

Therefore, we can apply case 2 of the Master Theorem, and so,

$$T(n) = \theta(n)$$

(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 $i \leq \lfloor n/2 \rfloor$.

Let's assume we have a perfect tree of depth k. This tree will have $2^{k+1} - 1$ nodes. We can also say that, up to level $k - 1$, the tree is perfect and has $2^k - 1$ nodes. Additionally, we can also say that, at the last level containing only leaves, there are $n - 2^k + 1$ nodes. Finally, we can say that each leaf on the last level will have a parent node. A pair of nodes will also share the same parent, and every node will share a parent with another node. Finally, out of the $2^k - 1$ nodes at level $k - 1$, there are $\lceil \frac{n-2^k+1}{2} \rceil$ parents and the rest are leaves, or $2^{k-1} - \lceil \frac{n-2^k+1}{2} \rceil$.

Therefore, we can say that the total amount of leaves is $n - 2^k + 1 + 2^{k-1} - \lceil \frac{n-2^k+1}{2} \rceil$ or, simplified, $\lceil \frac{n}{2} \rceil$