CS 212 **DUE: Tuesday, May 14 at 11:59 PM**
Spring 2019
The last assignment
5/7/2019 **10 points**

Introduction: This assignment will give you practice with analysis of algorithms.

This is an individual effort homework assignment. You must write up your solutions in LaTeX. Use the `a9.tex` template that I provide and be sure to replace each "Put your answer for ___ here." with your answers but leave everything else alone. Your solutions must be written up in a clear, concise and rigorous manner.

When you are done, zip up your .TEX file and corresponding .PDF file. Upload your .ZIP file to the **a9** dropbox on d2l. After you have uploaded the file, double-check to ensure your file was uploaded correctly. It is your responsibility to ensure your submission was done correctly. Assignments that are not uploaded correctly are worth 0 points.

Consider the following "`search`" algorithm, which returns the index of `x` in `a` if `x` exists in `a` and `-1` otherwise:

```
static int search(int[] a, int x) {
  for (int h = a.length; h > 0; h /= 2)
    for (int i = a.length - 1; i >= 0; i -= h)
      if (a[i] == x)
        return i;
  return -1;
```

The idea is that, you can pass in an `int` array `a` to search, along with a target value `x`, for which to search, and the `search` algorithm will either return a valid index, at which `x` occurs in `a`, or `-1`, if `x` does not occur in `a`. You could do a simple test with the following code:

```
public static void main(String args[]) {
   int[] a = { 3, 14, 1, 5, 9, 2, 6 };
   int x = 1, /* x = 99 */
       i = search(a, x);
   System.out.println(i);
}
```

The following parts will take you through a running time analysis of `search`. This is a good example of the type of problem you may see early on in an Algorithms class (e.g., CS 321).

1. (1 point) Give a good definition for N, i.e., the *size* of the input to `search`.

   $N \equiv$ the length (size) of the array to search

2. (6 points) In this question, you will do a worst-case running time analysis of `search`. Your goal will be to derive tight bounds, in terms of Θ, on the worst-case running time of `search`.

   (a) (1 point) First things first, give a clear yet concise description of a worst-case input to `search`. Don't forget that this input must be of size N.

   The worst-case input to `search` is an array of size N which does not contain the element x.

(b) (2 points) Give an asymptotic lower bound, using Ω notation, for the worst-case running time of `search` that holds for *at least one* input of size N. Be sure to specify exactly what the input is and then justify why your lower bound holds for it. Hint: see your answer to question 2a.

A bad asymptotic lower bound would be: $\Omega(1)$. The reason this is a bad answer, is because, for *any* input, `search` (or pretty much *any* algorithm) must execute at least one simple operation.

First, let's derive an expression that approximates the run time of `search`. We can see that the outer for loop runs in approximately $\log_2 N$ time, because each time the loop runs, it halves its control variable, which is initially set to be the size of the input N, and is allowed to go to a minimum of 1. The inner for loop runs in N time because its control variable is initially set at $N-1$, but it is allowed to go to zero, unlike the outer for loop. For all intents and purposes, h is a constant with respect to the inner for loop, therefore a constant amount is being subtracted from i as it searches through the array. This makes its run time N. Since this is the worst case input, it will always do the if statement, which is a constant amount of work (call it 1), and the return inside of that will never execute. This means that the return at the end will execute, which is a constant time operation. This gives us the equation: $T(N) = 1 \cdot N\log_2 N + 1$. We can now derive constants c and N_0 such that $1 \cdot N\log_2 N + 1 = \Omega(N\log N)$:

$$1 \cdot N\log_2 N + 1 \geq 1 \cdot N\log_2 N + 1$$
$$N\log_2 N + 1 \geq N\log_2 N$$

Now if we take $c = 1$ and $N_0 = 2$, we get $3 \geq 2$ for the base case. We know this is true for all values of N within the domain of each function, because the left hand side is always 1 greater than the right hand side. Therefore, our asymptotic lower bound is $\Omega(N\log N)$.

(c) (2 points) Give an asymptotic upper bound, using O notation, for the worst-case running time of `search` that holds for *all* inputs of size N and that matches the lower bound you gave in the previous part. Justify why your upper bound holds for all inputs.

Here, $O\left(N^2\right)$ would be a valid answer, but it's probably too big of an upper-bound. It's valid, because, for any input, the outer loop does at most N iterations, and, for each iteration of the outer loop, the inner loop does at most N iterations, and in each iteration of the inner loop, $O(1)$ simple operations are executed. So, in total, `search`, for any input, executes at most $O\left(N^2\right)$ simple operations.

So, basically, try to figure out a *lower* upper bound that is valid for all inputs of size N that you could pass to `search`.

Now we'll just take the equation from above and use it to find c and N_0 such that $1 \cdot N\log_2 N + 1 = O(N\log N)$:

$$\begin{aligned} 1 \cdot N\log_2 N + 1 &\leq 1 \cdot N\log_2 N + 1 \\ N\log_2 N + 1 &\leq N\log_2 N + 1 \\ &\leq N\log_2 N + N\log_2 N \\ &\leq 2N\log_2 N \end{aligned}$$

Now let's let $c = 2$ and $N_0 = 2$. This gives us $3 \leq 4$. We also know that $N\log_2 N + 1 \leq 2N\log_2 N$ for $N \geq N_0$ because $N\log_2 N \geq 1$ for $N \geq N_0$. Therefore, our asymptotic upper bound is $O(N\log_2 N)$.

(d) (1 point) Since the upper and lower bounds you gave in 2b and 2c, respectively, match (or, at least they're supposed to match), then what can you conclude about the asymptotic running time of `search`?

Since the asymptotic upper and lower bounds are the same, that means we can assign the asymptotic tight bound of $T(N) = \Theta(N \log_2 N)$ to our algorithm.

3. (3 points) In this question, you will do a best-case running time analysis of `search`. Here, we will define the *best-case* input to be the input of size N that minimizes $T(N)$, where $T(N) =$ the maximum number of simple operations performed by the algorithm for some input of size N.

   (a) (1 point) First things first, give a clear yet concise description of a best-case input to `search`. Don't forget that this input must be of size N.

   The best case input to `search` is an array of size N where the element x is at the end of the array (index $N-1$). This is because `search` always looks at the last index of the array for element x first.

(b) (1 point) Give an asymptotic upper bound, using O notation, for the running time of `search` that holds for *some* input of size N. Be sure to specify exactly what the input is and then justify why your upper bound holds for it. Hint: see your answer to question 3a.

Given the best case input, the outer for loop does a constant amount of work initializing the control variable, the inner for loop does a constant amount of work initializing the control variable, the if statement inside the inner for loop does a constant amount of work checking to see if the last element in the array is x, (which is true for the best case input), and finally, the return statement inside of the if statement does a constant amount of work when it returns the value i. Therefore, since `search` performs some constant amount of work given the best case input, $T(N) = O(1)$.

(c) (1 point) Finally, give an asymptotic lower bound, using Ω notation, for the running time of `search` that holds for *all* inputs of size N. Justify why your lower bound holds for all inputs.

Since all algorithms run in at least constant time, and our asymptotic upper bound is $O(1)$, the asymptotic lower bound of $T(N)$ is $\Omega(1)$ because our asymptotic lower bound cannot be greater than our asymptotic upper bound.