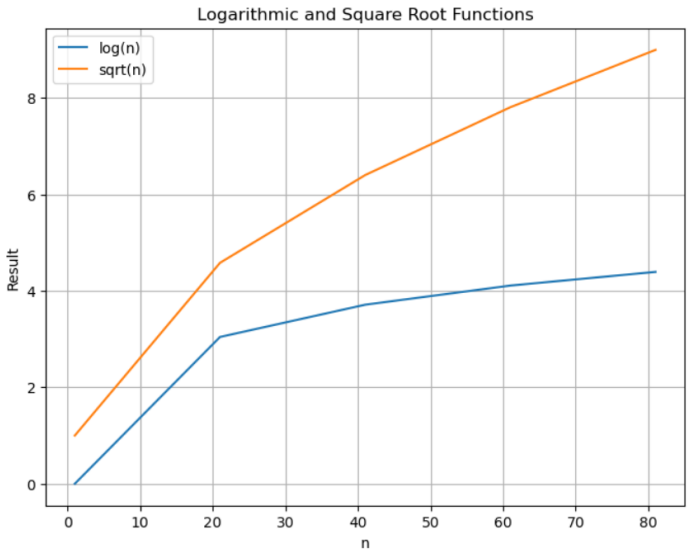
Tahir Emre Semiz – 20200808058

Mod 4

1b –



The Big O Notation is represented as the upper bound of a function. It is stated in the question that the log(n) is at the upper limit of $\sqrt{n}$, but as you can see in the graph above, the $\sqrt{n}$ function is a larger function than the log(n) function. Therefore, the function log(n) cannot be an upper bound of $\sqrt{n}$. So the answer is false.

4a –

We can solve this question briefly with master theorem. To remember, the representation and conditions of master theorem are as follows:

T(n) = aT(n/b) + f(n)

a ≥ 1 , b > 1 and also f(n) is an asymptotic positive function.

Then, according to the values found, whichever of the 3 cases satisfies the condition is applied and the result is obtained.

In this case, if we examine the function in the question according to the formula, the values of a, b and f(n) are as follows:

a = 4, b = 3

f(n) = $\Theta(n^k \log^p n)$ according to this;

k = 1, p = 0

We use case 1 according to the values we found.

Case 1: if $\log_b a > k$ then $\Theta(n^{\log_b a})$

$\log_b a = \log_3 4$

$\log_3 4 = 1.26$

So result is $\boldsymbol{\Theta(n^{1.26})}$

5b –

If we go line by line:

r:=0: 1 assignment

for i:=1 to n do: n assignments

for j:=1 to i do: n(n+1)/2 assignments

for k:= j to i + j do: n(n+1)(n+2)/3 assignments

r := r + 1: same as previous line

In total: $1+n+(n(n+1)/2)+(n(n+1)(n+2)/3)\cdot 2 = (n+1)(n+2)(4n+3)/6$

As a result: $O(n^3)$

6a –

First, let's explain the elements in the question:

S_1 - less than the pivot value

S_2 - those equal to the pivot value

S_3 - those greater than the pivot value

In the base case, that is, T(1), it will be equal to T(1) = 0 since a single element does not need to be sorted.

Since it will be solved with master theorem, we create the recurrence relation. It will look like this:

$$\mathbf{T(n)= T(|S1|) + T(|S3|)}$$

If we consider the individual elements of S_1, S_2 and S_3, in the worst case scenario, S_1 and S_3 have the value of n/2 because they represent parts smaller and larger than the pivot. During the pivot search process, an operation of (n - 1) value is performed.

$$\mathbf{T(n)= T(n/2) + T(n/2) + (n - 1)}$$

$$\mathbf{T(n)= 2T(n/2) + (n - 1)}$$

We will determine the values before master theorem:

$$\mathbf{a = 2, b = 2}$$

$f(n) = \Theta(n^k \log^p n)$ according to this;

$k = 1, p = 0$

$\log_b a = 1$

$\log_b a = k$

So Case 2 is applied. If $p > -1$, $n^k \log^{p+1} n$ is applied. The result is found as **$\Theta(n \log n)$.**

7 –

Analysis for comparison quantity:

There are 2 comparisons due to the if-else statements in the first two lines.

There is 1 if-else and 2 if inside the for loop. Since the for keyword will be returned n/2 times, it becomes 4*(n/2).

In total, we reach the value **2 + 4 * (n/2).**

We can do it with the divide and conquer method. The code looks like this:

```
public static int[] findMaxMin(int[] arr, int l, int r) {
    if (l == r) {
        return new int[]{arr[l], arr[l]};
    }
    else if (l + 1 == r) {
        return new int[]{Math.max(arr[l], arr[r]), Math.min(arr[l], arr[r])};
    }
    else {
        int mid = (l + r) / 2;
        int[] lResult = findMaxMin(arr, l, mid);
        int[] rResult = findMaxMin(arr, mid + 1, r);
        int max = Math.max(lResult[0], rResult[0]);
        int min = Math.min(lResult[1], rResult[1]);
        return new int[]{max, min};
    }
}
```

To explain the code line by line:

- For an array with a single element, we returned a base case 1 whose max and min values are equal to that element.

- Since we cannot divide and conquer an array with two elements, we returned the max and min values between them directly in base case 2.

- We proceed recursively by dividing the array into two parts, left and right, each time. After proceeding to the end, we use the functions in the math library to compare the largest and smallest elements of the parts on the left and right and obtain the minimum and maximum values of the entire array.