Ryan Pollack

rep127

I have not discussed this exam with anyone except the professor and the TAs of CS 344. I have not used any online resources during the exam except for those accessible from the Canvass website.

1. Part 1

$$T(n) \leq 64T(n/4) \ + n^2$$

Layer	Number of calls	Problem size	Non-recursive work
1	1	n	n^2
2	64	n/4	$64 * (\frac{n}{4})^2$
3	64^2	n/4^2	$64^2 * (\frac{n}{4^2})^2$
…	…	…	…
k	64^{k-1}	n/4^{k-1}	$64^{k-1} * (\frac{n}{4^{k-1}})^2$

$$64^{k-1} * \left(\frac{n}{4^{k-1}}\right)^2 = \frac{n^2}{4^{(k-1)^2}} * 4^{(k-1)^3} = n^2 * 4^{(k-1)}$$

$$64^{k-1} = 4^{k-1} * 4^{k-1} * 4^{k-1} = 4^{3(k-1)}$$

The non-recursive work on the kth level is $n^2 * 4^{(k-1)}$

$\sum_{k=1}^{\log n+} \ n^2 * 4^{(k-1)} = n^2 * (1 + 4 + 16 + 64 + \cdots)$ geometric series

$$\sum_{k=1}^{\log n+} n^2 * 4^{(k-1)} = n^2 * \sum_{k=1}^{\log n+} 4^{(k-1)}$$

$$= n^2 * \left(\frac{4^{\log n+} \ - 1}{4 - 1}\right)$$

$$= n^2 * \left(\frac{4}{3} * \frac{4^{\log n}}{3} - \frac{4}{3}\right)$$

$$= \frac{4}{3}n^3 - n^2$$

$$T(n) = n^3$$

1. Part 2

$$T(n) \leq 2T\left(\frac{n}{2}\right) + 3T\left(\frac{n}{3}\right) + n^2$$

Considering, $T(n) = O(n^2)$

Assume $T(n) \leq cn^2$ where some c>0

Base case: c = 1, so $1n^2 = n^2$

So,

$$T(n) \leq 2T\left(\frac{n}{2}\right) + 3T\left(\frac{n}{3}\right) + n^2$$

$$T(n) \leq 2\left(c * \left(\frac{n}{2}\right)^2\right) + 3\left(c * \left(\frac{n}{3}\right)^2\right) + n^2$$

$$T(n) \leq \frac{c * n^2}{2} + \frac{c * n^2}{3} + n^2$$

$$T(n) \leq \left(\frac{5c}{6} + 1\right)n^2$$

$$T(n) \leq cn^2$$

$$O(n^2)$$

2.

I don’t know

3.

$$T(n) \leq \mathrm{T}\left(\frac{\mathrm{n}}{13}\right) + \mathrm{T}\left(\frac{19\mathrm{n}}{26}\right) + O(n)$$

The T(n/13) part is derived from the fact that when we run the recursive SELECT method, we are running it in the 13 different chunks which fills the new array of size n/13 that holds the mini median from each of the 13 chunks. For each of these individual chunks, picking out the mini median is O(1) time and there is n/13 groups so T(n/13).

The O(n) is derived from partitioning around the chosen m. It goes one by one and compares each chunk median with the m and positions said element on the left side (lesser than m) or right side (greater than m) of m based on the comparison. (Dividing the array into 13 chunks and creating the secondary array of length n/13 also take O(n) time.)

The T(19n/26) is derived based from the worst-case of how far away from the center m can be. To prove this, we can start by saying by definition, half of the mini medians will be lesser than m (the medians of the medians) and half will be larger than m, so (n/13)/2 = n/26 for each lesser half (as well as larger half but we're proving this with small). Looking into each lesser half, we know the median of that chunk is smaller than m. However, we also know since it's the median of the chunk, at least half of the elements in there will also be smaller than m since half are smaller than the mini medium. In the lesser chunks with 13, that is the median + 6 other elements are bare minimum, so 7 * (n/26) = 7n/26. With this same concept applied on the larger side, we know m can at most be 19n/26, where at least 7 elements are bigger in each larger chunk.

4.

```
Set closest = int_Max
Set answer = 0
Initialize D_B
For i=0 to n-1
        D_B.insert(B[i].value, B[i])
End for
For j=0 to n=1
        Set diff = abs_value(A[j] - D_B.search(A[j].value))
        If diff < closest
                closest = diff
                answer = A[j].value
        End if
End for
Return answer
```

5.

```
ThresholdSum(A, T)
        Set the ints value, largest, and count to 0
        For i=0 to n-1          //deals with no solution
                Value += A[i]
        End for
        If value ≤ T
                Return “no solution”
        End if

        ThresholdSumRecursive(A, T)
                For i=0 to n-1
                        If largest < A[i]
                                Set largest = A[i]
                        End if
                End for
                count++
                result += largest
                Remove A[i] that ended as largest from array
                If(result > T)
                        Return count
                Else
                        ThresholdSumRecursive(A, T)
                End ifelse
```