Assignment2

Prashanth Vangari

Student Id: 11645119

Section: 004

1]

Algorithm A :

Since the algorithm is getting divided into five subproblems of half size which is $\frac{n}{2}$ and combining the solution is taking linear time which is 'n'. We have

T(n)=5$T(\frac{n}{2})$+n

It is in the form of a$T(\frac{n}{b})$+f(n)

Here a=5, b=2, f(n)=n, $n^{\log_b a}$=$n^{\log_2 5}$ => $n^{2.321}$

Compare with $n^{\log_2 5}$ with f(n)

Case 1:

f(n)= $O(n^{\log_b a-\varepsilon})$ for some ε>0 then T(n)= $\Theta(n^{\log_b a})$

⇨ $n^{\log_2 5-\varepsilon}$

T(n)= $\Theta(n^{\log_2 5})$

⇨ **T(n)= $\Theta(n^{2.32})$**

Time complexity of algorithm A is $\Theta(n^{2.32})$

Algorithm B:

Since the algorithm is solving problems of size n by recursively solving two subproblems of size n - 1 and then combining the solutions in constant time, we have

T(n)=2T(n-1)+c

T(n-1)=2T(n-1-1)+c

⇨ 2T(n-2)+c

$T(n-2)=2T(n-3)+c$

$T(n)=2[2T(n-2)+c]+c$

- ⇨ $2^2T(n-2)+2c+c$
- ⇨ $2^2[2T(n-3)+c]+2c+c$
- ⇨ $2^3T(n-3)+4c+2c+c$

$T(k)=2^kT(n-k)+c.2^{k-1}+c.2^{k-2}+\ldots+c$

- ⇨ $2^kT(n-k)+c[2^{k-1}+2^{k-2}+\ldots+1]$
- ⇨ $2^kT(n-k)+c\sum_{k=1}^{k} 2^{k-1}$
- ⇨ $2^kT(n-k)+c[\frac{2^{k-1+1}-1}{2-1}]$
- ⇨ $2^kT(n-k)+c[2^k-1]$

Let $n-k=0 \Rightarrow n=k$

- ⇨ $2^kT(0)+c.2^n$
- ⇨ $2^n(1)+c.2^n$
- ⇨ $2^n(1+c)$
- ⇨ $\Theta(2^n)$

Time complexity of Algorithm B is $\Theta(2^n)$

Algorithm C:

Since the algorithm C solves the problems of size n by dividing them into nine subproblems of size n/3, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time, we have

$T(n)=9T(\frac{n}{3})+n^2$

Here a=9, b=3, $f(n)=n^2$, $n^{\log_b a}=n^{\log_3 9}=n^2$

Compare with $n^{\log_3 9}$ with f(n)

Case2:

$f(n)=\Theta(n^{\log_b a})$, then $T(n)=\Theta(n^{\log_b a}.lgn)$

$T(n)=\Theta(n^{\log_b a}.lgn)$

- ⇨ $\Theta(n^{\log_b a}.\mathrm{lgn})$
- ⇨ $\Theta(n^{\log_3 9}.\mathrm{lgn})$
- ⇨ $\Theta(n^2.\mathrm{lgn})$

Solution: The running time of Algorithm A is $\Theta(n^{2.32})$, Algorithm B's running time is $\Theta(2^n)$ and Algorithm C's running time is $\Theta(n^2.\text{lgn})$.

I would choose Algorithm C because it has lower order term and has lower time complexity compared to Algorithm A and Algorithm B

2]

1. Pseudo code:

1. If the size of the sequence is 2, then the maximum of the two elements becomes the maximum number and the minimum of those two numbers becomes the minimum.
2. Recursively call the same function with first half of the array to find the maximum and the minimum.
3. Similarly call the second half of the array to find the maximum and minimum of the remaining sequence.
4. Now return the minimum and maximum of the both left sequence and right sequence.

```
Max_Min(sequence, low, high)
        if(n==2) //n is size of the sequence
return(min(sequence[0],sequence[1]),max(sequence[0],sequence[1])
        else
                mid = (low+high)/2
                (minLeft, maxLeft) = Max_Min(sequence, low, mid)
                (minRight, maxRight) = Max_Min(sequence, mid+1, high)
        return(min(minLeft, minRight), max(maxLeft,maxRight))
```

2] Time complexity

Here the sequence is getting two times divided into two halves and then combining the solution in a constant time of 2. So we have

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

$T(\frac{n}{2}) = 2T\left(\frac{n}{4}\right) + 2$

$T(\frac{n}{4}) =2T(\frac{n}{8}) + 2$

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

- ⇨ $2[2T\left(\frac{n}{4}\right) + 2]+2$
- ⇨ $2^2T\left(\frac{n}{4}\right) + 4 + 2$
- ⇨ $2^2[2T(\frac{n}{8}) + 2]+4+2$
- ⇨ $2^3T(\frac{n}{8}) + 8 + 4 + 2$

$T(k)=2^kT(\frac{n}{2^k}) +2^k+2^{k-1}+2^{k-2}+.....+2$

- ⇨ $2^kT(\frac{n}{2^k}) + \sum_{k=1}^{k} 2^k$
- ⇨ $2^kT(\frac{n}{2^k}) + \frac{2^k-1}{2-1}$

let $n=2^k$

- ⇨ $n.T(1)+(n-1)$

let $T(1)=1$

- ⇨ $n+n-1 => 2n+1$
- ⇨ $O(n)$

Solution: The time complexity of the algorithm is $O(n)$

3]

Given,

$T(n)=\{$

$c = \text{if } n\leq 2,$

$T(n-2)+n \ \text{ otherwise}$

$\}$

Case 1: Consider n is even.

$T(1)=c$

$T(2)=c$

$T(n)=T(n-2)+n$ for $n>2$

T(n-2)=T(n-2-2)+(n-2)

- ⇨ T(n-4)+(n-2)

T(n-4)= T(n-6)+(n-4)

T(n-6)=T(n-8)+(n-8)

T(n)= T(n-2)+n

- ⇨ T(n-4)+(n-2)+n
- ⇨ T(n-6)+(n-4)+(n-2)+n
- ⇨ T(n-8)+(n-6) +(n-4)+(n-2)+n

After k iterations

T(n)=T(n-2k)+(n-2(k-1))+(n-2(k-2))+ (n-2(k-3))+…….+n

- ⇨ T(n-2k)+(n-2k+2)+(n-2k+4)+(n-2k+6)+………..+n

Let n-2k=0 => n=2k

- ⇨ T(n-n)+(0+2)+(0+4)+(0+6)+…..+n
- ⇨ T(0)+2+4+6+….+n
- ⇨ T(0)+2[1+2+3…n/2]
- ⇨ c+2[$\frac{\frac{n}{2}(\frac{n}{2}+1)}{2}$]
- ⇨ c+($\frac{n^2}{4} + \frac{n}{2}$)
- ⇨ c+$\frac{n^2}{4} + \frac{n}{2}$

By ignoring the lower order terms and constant c we get

- ⇨ n^2
- ⇨ $O(n^2)$

Asymptotic upper bound for T(n) is $O(n^2)$ when n is even.

Case 2: Consider n is odd

T(1)=c

T(2)=c

T(n)=T(n-2)+n

T(n-2)= T(n-2-2)+(n-2)

- ⇨ T(n-4)+(n-2)

T(n-4)= T(n-6)+(n-4)

T(n-6)=T(n-8)+(n-6)

T(n)=T(n-2)+n

- ⇨ T(n-4)+(n-2)+n
- ⇨ T(n-6)+(n-4)+(n-2)+n
- ⇨ T(n-8)+(n-6)+ (n-4)+(n-2)+n

So the k^{th} term becomes

T(n)=T(n-2k)+(n-2(k-1))+ (n-2(k-2))+……+n

Since n is odd, n=2k+1

- ⇨ T(2k+1-2k)+(2k+1-2k+2)+(2k+1-2k+4)+….+n
- ⇨ T(1)+3+5+7+…n

The above sequence forms an arithmetic progression with a=3, d=2

Therefore summation becomes $\frac{n}{2}[2a + (n-1)d]$

- ⇨ T(1)+$\frac{n}{2}[2 * 3 + (n-1)2]$

n=2k+1 => k=(n-1)/2

- ⇨ c+$\frac{n-1}{2}[6 + [\frac{(n-3)}{2}]2]$
- ⇨ $c + \frac{n-1}{2}[n + 3]$
- ⇨ c+$\frac{n^2-n+6}{2}$
- ⇨ c+$\frac{n^2}{2} - \frac{n}{2} + 3$
- ⇨ By ignoring the lower order terms and the constant 'c' we get
- ⇨ O(n^2)

Asymptotic upper bound for T(n) is O(n^2) when n is odd.

Solution: The algorithm has O(n^2) time complexity for both odd and even number. **Therefore the time complexity of the algorithm is O(n^2) .**