Assignment

Name : Faria Huq

Roll : 1505052

1.

From Master Theorem We find that ,

T(n) = a*T(n/b) + f(n)

For Algorithm A , T(n) = 5*T(n/2) + n

a =5 , b= 2 , f(n) = n.

Here , $n^{\log_2 5}$ ≈ nΛ(2.32)

F(n) = O(nΛ(2.32-ε) where ε≈ 1.32 which follows the first case .

So ,T(n) = Θ($n^{\log_2 5}$) = O($n^{\log_2 5}$)

For Algorithm B ,

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

Following Recursion Tree ,

T(n) = 1 + 2^l + 2Λ2 + 2Λ3 + 2^4 + …. + 2^n

= 2^n - 1

So , $T(n) = \Theta(2^n - 1)$

For Algorithm C ,

T(n) = 9*T(n/3) + nΛ2

a =9 , b= 3 , f(n) = nΛ2

$n^{\log_3 9}$ = nΛ2 = F(n)

So , T(n) = Θ((nΛ2)*logn) = O((nΛ2)*logn)

I will choose Algorithm C as It is more efficient .

2 .

A)

Knowing majority elements (m) of A1 and A2 are not enough only , we need to know their repetition numbers (r)'s as well.

Step 1 : Divide the array in half , And call the function recursively for each half . Each will return their m and r . Stop recursive call when low == high .

Mid = (low+high)/2;

Step 2 : Compare the majority element and r of each half . If the majority elements match , we will return the r and summation of its repetition numbers in the divisions . Else , we will check their repetition number in each half , if any of these is more than (high-low)/2 , then return the element with corresponding repetition number ,else return zero . If the subarray has only one element return the value with r=1.

B)

There is a total of N elements , we have $\lceil n/2 \rceil$ pairs . From each pair , We can one or zero elements , so after each iteration , we are left with

n^{l} elements where $0 \leq n^{l} \leq \lceil n/2 \rceil$.

If every element is unique , Each pair returns a single element comprising $\lceil n/2 \rceil$ elements in total . so , there can be at most $\lceil n/2 \rceil$ elements after each iteration .

Suppose , A has a majority element r , so its repetition number , m = $n/2 + \partial$ where $1 \leq \partial \leq n/2$ and the total number of its other elements is ε ($\varepsilon < m$) . There are at most $\lceil \varepsilon/2 \rceil$ pairs where the return values are not equal to m . As , $\lceil \varepsilon/2 \rceil$ is always less than $\lceil m/2 \rceil$ and $n^{l} = \lceil \varepsilon/2 \rceil + \lceil n/2 \rceil$.

So , A^{l} has a majority element only if A does .

We will repeat the same procedure on A^{l} and finish when there is only one pair left . If the elements in this pair are unique , we have a majority element .

3)

In greedy algorithm method , We use minheap to generate the Huffman Tree . We extract two most minimum values from the minheap and create an equivalent node where the removed two elements are present as children to the newly created node until there is only one element left in the heap and that element is the root of the generated Huffman tree . As We extracted two most minimum values at each level , the smallest values will be silblings and remain at the most bottom level , and gradually building upwards , so it ensures an optimized solution . If we can ensure that after each insertion , We have an optimal prefix code for the generated huffman tree , we can proof its optimization .

Let , C be a given alphabet with frequency c.freq for each character cεC and T be its tree where two child x and y exists such that x.freq and y.freq are most minimum . We generate T' removing x and y and inserting z so that z.freq = x.freq + y.freq .

For each character c ε C – {x,y} , we have $d^{t}(c) = d^{t'}(c)$ and hence , have c.freq.$d^{t}(c)$ = c.freq.$d^{t'}(c)$

Since , $d^{t}(x) = d^{t}(y) = d^{t}(z) + 1$, we have

x.freq. $d^{t}(x)$ + y.freq. $d^{t}(y)$ = (x.freq + y.freq) ($d^{t}(z)$+1)

= z.freq. $d^{t}(z)$ + (x.freq + y.freq)

From which we conclude ,

B(T) = B(T') + (x.freq + y.freq) ;

Or , B(T') = B(T) - (x.freq + y.freq) ;

As , B(T) is unique , B(T') is also unique so that for no other tree exists such that B(T'')< B(T').

Thus , T' must represent an optimal prefix code for alphabet   C , so Huffman code generates the optimal solution .

CHAR-FREQ-LENGTH-CODE

e : 102 - 1 - 0

a : 68 - 2 - 10

d : 35 - 3 - 110

c : 26 - 4 - 1110

b : 13 - 5 - 11110

f : 18 - 5 – 11111

4 . a) n = request number

s[1….n] = starting time of each request

f[1….n] = finishing time of each request

v[1…n] = values of each request

sort requests by finishing time , $f1 \leq f2 \ldots.. \leq fn$ (*running time = O(nlogn)*)

Compute p[1], p[2], …, p[n] where p (j) = largest index i < j such that job i is compatible with j (*running time = O(nlogn)*)

```
for j = 1 to n
M[j] ← empty.
M[0] ← 0.
```

M-Compute-Opt(j) (*running time = O(n)*)

```
if M[j] is empty
```

M[j] = max(v[j] + M-Compute-Opt(p[j]), M-Compute-Opt(j – 1)).

return M[j].

Find-Solution(j) *(running time = O(nlogn))*

```
if j = 0 return Ø.
else if (v[j] + M[p[j]] > M[j–1])
return { j } ∪ Find-Solution(p[j]).
else
return Find-Solution(j–1).
```

Overall running time is *O(nlogn)* .

6. Let,

Rel(i,b) = most reliable configuration of machines 1 to i available within budget b

As we are doing the performance sequentially , we run a loop from 1 to n and after buying m^i copies of c^i machines we will be left with

B = B - (m^i * c^i) , but We can not buy unless B>0 .

If there is still budget left , we can buy more machines . for this , We try all possibilities m of machine of type i to buy machines which can vary from 1 to [B/ c^i] . as i>0 we need finish recursive call when i = 0 .

Rel[n][b]→all zeros

$$\mathrm{Rel}[n][b] = \begin{cases} 0 \quad \text{when } b<0 \text{ or } b=0 \text{ but } i>0 \\ 1 \quad \text{when } b \geq 0 \text{ , } i = 0 \\ \mathrm{Max}(\ \mathrm{Rel}\ [n-1][B-(m^{i} * c^{i}) * (1-(1-r^{i})^{m}]) \quad \text{where } 1 \leq m \leq [B/\ c^{i}] \end{cases}$$