Tutorial 2, Design and Analysis of Algorithms, 2020

1. The sets A and B have n elements each given in the form of sorted arrays. Design $O(n)$ algorithms to compute $A \cup B$ and $A \cap B$.

To compute AUB:

UNION(A,B)

1) la = length(A)
2) lb = length(B)
3) ia = ib = 0 // assume 0 indexed arrays
4) while ia < la AND ib < lb
   4.1) if (A[ia] < B[ib])
      4.1a) result.append(A[ia])
      4.1b) ia = ia+1
   4.2) else if (A[ia] > B[ib])
      4.2a) result.append(B[ib])
      4.2b) ib = ib+1
   4.3) else
      4.3a) result.append(A[ia])
      4.3b) ~~result~~ ia = ia+1
      4.3c) ib = ib+1
5) while ia < la
   5.1) result.append(A[ia])
   5.2) ia = ia+1
6) while ib < lb
   6.1) result.append(B[ib])
   6.2) ib = ib+1
7) return result.

// add an extra line initializing result in the exam.

Correctness:

Consider an element in the array A (without loss of generality). Let its index in A be i_{ea}. When we compare it with the largest element in B, there are two possibilities:

1) $A[i_{ea}]$ is larger than the largest element in B ($B[lb-1]$).
Then, there will have been a stage in the algorithm in the first while loop (4) where
$ia \leq i_{ea}$, $ib == lb-1$.
In this stage, step 4.2 is executed and $ib = lb$.
Then the loop condition in step 4 is not satisfied and the first while loop is exited.
Since $ia < la$ and $ia \leq i_{ea} < la$, the while loop in step 5 is entered. Since ia only changes by 1, when this loop exits, $ia == la$. $\therefore ia > i_{ea}$.
Since ia only increments by 1, it must have been equal to i_{ea} at some time. Thus, at some point, $ia == i_{ea}$ and then the loop in step 5 would have been entered ($i_{ea} < la$) and $A[i_{ea}]$ would have been appended to result. Since A & B are sorted, the previous element in result would have been either $A[i_{ea}-1]$ or $B[lb-1]$, both of which are lesser than $A[i_{ea}]$.

2) $A[i_{ea}]$ is lesser than or equal to $B[lb-1]$
In this case, there will have been an i, $0 \leq i < lb$, such that either $B[i-1] < A[i_{ea}] \leq B[i]$ or $A[i_{ea}] = B[i]$
~~or $A[i_{ea}] = B[0]$~~ (assume for this algorithm that $B[-1] = -\infty$ & $A[-1] = -\infty$)

i) If $B[i-1] < A[i_{ea}] < B[i]$
~~As loop (4) was~~ As the loop in step 4 executes, $ib == i$ at some point and $i_{ea} == ia$. At this point, 4.1 will be entered and $A[i_{ea}]$ will be appended to result.

ii) If $A[i_{ea}] == B[i]$
As the loop in step 4 executes, at some point $ib == i$ and $i_{ea} == ia$. At this point, 4.3 will be entered and this element will be added once to result.

Complexity
Steps 1-3 are all O(1).
Steps 1,2 are O(n) (where n is the larger of the array sizes)
Step 3 is O(1).
Since steps (4.1), 4.2, 4.3 are all O(1) (assignment and appending an element), step 4 = O(n)O(1) = O(n).
Steps 5.1 and 5.2 are O(1), step 5 = O(n).
Similarly step 6 = O(n)
Finally, step 7 is O(1).
∴ Total complexity = O(n) + O(1) + O(n) + O(n) + O(n) + O(1)
= O(n).

For intersection, remove steps (4.1a), (4.2a), (5) and (6).
Complexity is still the same (remove contributions of above steps).
For correctness, show that an element in A will only be appended to result if it is equal to some element in B.

2. Consider two sets A and B, each having n integers in the range from 0 to $10n$. We wish to compute the Cartesian sum of A and B, defined by
$C = \{x + y \mid x \in A \wedge y \in B\}$
Note that the integers in C are in the range from 0 to $20n$. We want to find the elements of C and the number of times each element of C is realized as a sum of elements in A and B. Show how to solve the problem in $O(n \log n)$ time.

The idea is to frame the problem in terms of polynomial multiplication as follows:

Consider the polynomials $a(x) = \sum_{i \in A} x^i$ and $b(x) = \sum_{j \in B} x^j$

find their product $c(x) = a(x)\,b(x)$
any term $c_k x^k$ in $c(x)$ will be because of some terms x^{i_1}, x^{j_1}; x^{i_2}, x^{j_2}; x^{i_3}, x^{j_3}; ... ; x^{i_k}, x^{j_k} in $a(x)$ and $b(x)$ respectively, where
(Expand in detail if asked).

$i_1 + j_1 = i_2 + j_2 = \ldots = i_{c_k} + j_{c_k} = k$

But $x^{i_1} \in A(x)$ iff $i_1 \in A$ and $x^{j_1} \in b(x)$ iff $j_1 \in B$.
∴ The powers k of every term $c_k x^k$ in $c(x)$ are the elements of C and the coefficients c_k of these terms is the number of time k occurs as a sum.

This is for showing correctness.

The algorithm :-
Cartesian Sum (a, b)
1) n = length(a) ia = ib = 0. // make this a separate step
2) for i = 0 to $2^{\lceil \log_2(2n+2) \rceil}$
2.1) if (i == a[ia]) ∧ (ia < n) // reverse the order of ∧ operands in exam
2.1a) pb.append(i)
2.1b) ia = ia + 1
2.2) else
2.2a) pa.append(0)
2.3) if (i == b[ib]) ∧ (ib < n) // reverse order of ∧ operands in exam.
2.3a) pb.append(1)
2.3b) ib = ib + 1
2.4) else
2.4a) pb.append(0).
3) tpa = R-FFT(pa) // Check CLRS & copy
4) tpb = R-FFT(pb)
5) for i = 0 to $2^{\lceil \log_2(2n+2) \rceil}$
5.1) tpc.append(tpa[i] * tpb[i])
6) pc = Inverse-FFT(tpc) // Check CLRS & copy.
7) for i = 0 to length(pc)
7.1) if (pc[i] >= 1) c.append(i)
8) return c, pc // to find out the number of times an element C_k in C occurs in the cartesian sum is pc[C_k].

For correctness, everything is straightforward but to show that $2^{\lceil \log_2(20n+2) \rceil}$ is $O(n)$. for this, let $k = 1, 2, \ldots$

$2^{k-1} < 20n+2 \leq 2^k$ → ①

Multiplying ① by 2, we get

$2^k < 40n+4 \leq 2^{k+1}$ → ②

Taking $\log_2$ on ①, we get

$k-1 \leq \log_2(20n+2) \leq k$.

$\Rightarrow \lceil \log_2(20n+2) \rceil = k. \Rightarrow 2^{\lceil \log_2(20n+2) \rceil} = 2^k$ → ③.

From ③ & ②, we have.

$40n+4 > 2^{\lceil \log_2(20n+2) \rceil}$

Since the array has n elements, we know that for $n \geq 1$

$44n \geq 40n+4 > 2^{\lceil \log_2(20n+2) \rceil}$

$\therefore 1 \leq 2^{\lceil \log_2(20n+2) \rceil} \leq 44n, \forall n \geq 1,$

$2^{\lceil \log_2(20n+2) \rceil} = O(n)$ //.

The rest of the steps add up to $O(n \lg n)$ (Perform this addition in the exam)

3. Arrange the following algorithms in increasing order of time complexity: *Karatsuba's Divide and Conquer Integer Multiplication Algorithm, Iterative Matrix Multiplication Algorithm, Strassen's Divide and Conquer Matrix Multiplication Algorithm.* Give proper reasons for your answer.

Given that complexity of
i) Karatsuba's algorithm $= O(n^{\log_2 3})$
ii) Iterative matrix multiplication $= O(n^3)$
iii) Strassen's divide and conquer matrix multiplication $= O(n^{\log_2 7})$

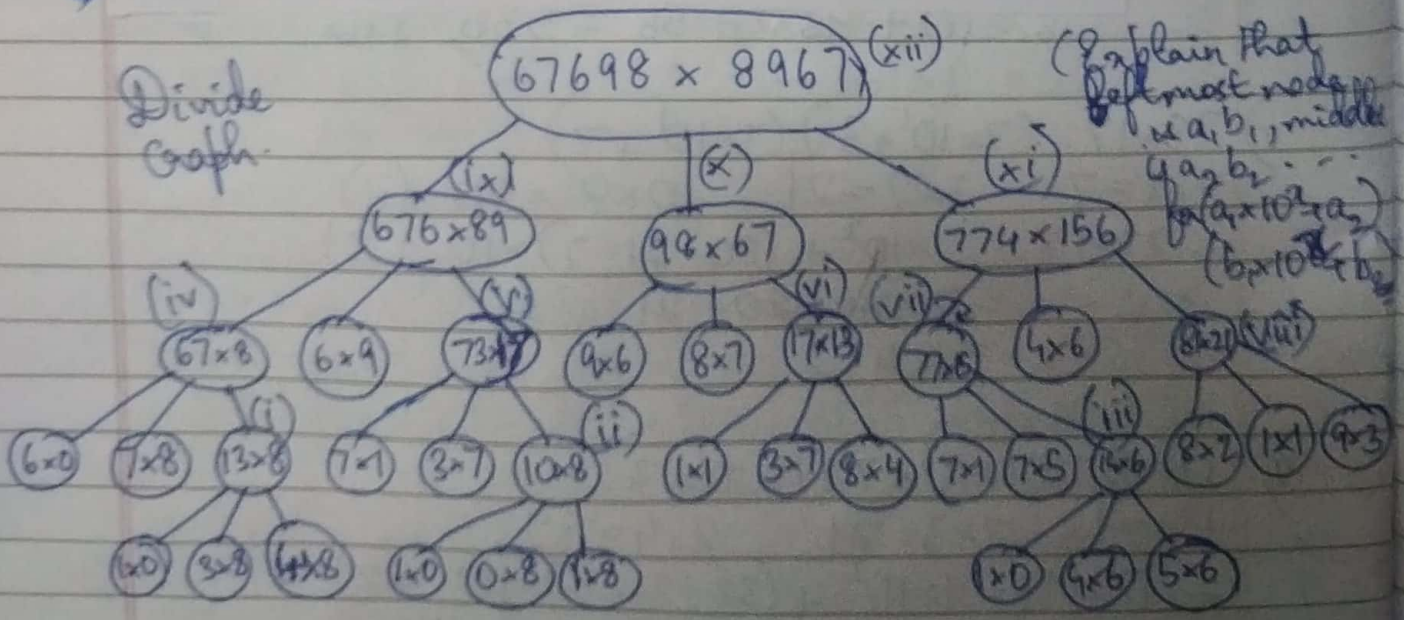
We have $3 < 7 < 8$
$\Rightarrow \log_2 3 < \log_2 7 < \log_2 8$
$\Rightarrow \log_2 3 < \log_2 7^2 < 3$

$\Rightarrow n^{\log_2 3} < n^{\log_2 7} < n^3$
$\Rightarrow O(n^{\log_2 3}) < O(n^{\log_2 7}) < O(n^3)$

∴ Karatsuba's algorithm < ~~Iterative matrix multip~~ Strassen's divide & conquer matrix multiplication < Iterative matrix multiplication.

(In the exam derive the above complexities).

4. Find 67698 × 8967 using *Karatsuba's Divide and Conquer Integer Multiplication Algorithm* showing the computations in a divide and conquer graph.



Describe the format for representation of each problem
Describe how 10^x was chosen (floor of $1/2$ (length of larger operand))

i) 13×8: $(1 \times 10^1 + 3)(0 \times 10^1 + 8)$

$1 \times 0 = 0$ $\quad 3 \times 8 = 24$ $\quad 4 \times 8 = 32$.

$\therefore 13 \times 8 = 0 \times 10^2 + (32 - 0 - 24) \times 10 + 24$

$= 0 + 80 + 24 = 104$ //

ii) 10×8: $(1 \times 10^1 + 0)(0 \times 10^1 + 8)$

$1 \times 0 = 0$ $\quad 0 \times 8 = 0$ $\quad 1 \times 8 = 8$.

$\therefore 10 \times 8 = 0 \times 10^2 + (8 - 0 - 0) \times 10 + 0$

$= 0 + 80 + 0 = 80$ //

iii) 14×6: $(1 \times 10^1 + 4)(0 \times 10^1 + 6)$

$1 \times 0 = 0$ $\quad 4 \times 6 = 24$ $\quad 5 \times 6 = 30$

$\therefore 14 \times 6 = 0 \times 10^2 + (30 - 24 - 0) \times 10 + 24$

$= 0 + 60 + 24 = 84$ //

iv) 67×8: $(6 \times 10^1 + 7)(0 \times 10^1 + 8)$

$6 \times 0 = 0$ $\quad 7 \times 8 = 56$ $\quad 13 \times 8 = 104$ (i)

$\therefore 67 \times 8 = 0 \times 10^2 + (104 - 56 - 0) \times 10 + 56$

$= 0 + 480 + 56 = 536$ //

v) 73×17: $(7 \times 10^1 + 3)(1 \times 10^1 + 7)$

$7 \times 1 = 7$ $\quad 3 \times 7 = 21$ $\quad 10 \times 8 = 80$ (ii)

$\therefore 73 \times 17 = 7 \times 10^2 + (80 - 21 - 7) \times 10 + 21$

$= 700 + 520 + 21$

$= 1241$ //

vi) 17×13: $(1 \times 10^1 + 7)(1 \times 10^1 + 3)$

$1 \times 1 = 1$ $\quad 7 \times 3 = 21$ $\quad 8 \times 4 = 32$.

$\therefore 17 \times 13 = 1 \times 10^2 + (32 - 21 - 1) \times 10 + 21$

$= 100 + 100 + 21 = 221$ //

vii) $77 \times 15 = (7 \times 10^1 + 7)(1 \times 10^1 + 5)$

$7 \times 1 = 7 \quad 7 \times 5 = 35 \quad 14 \times 6 = 84$ (iii)

$\therefore 77 \times 15 = 7 \times 10^2 + (84 - 35 - 7) \times 10 + 35$

$= 700 + 420 + 35$

$= 1155$

viii) $81 \times 21 = (8 \times 10^1 + 1)(2 \times 10^1 + 1)$

$8 \times 2 = 16 \quad 1 \times 1 = 1 \quad 9 \times 3 = 27$

$\therefore 81 \times 21 = 16 \times 10^2 + (27 - 16 - 1) \times 10 + 1$

$= 1600 + 100 + 1 = 1701$

ix) $676 \times 89 = (67 \times 10^1 + 6)(8 \times 10^1 + 9)$

$67 \times 8 = 536$ (iv) $\quad 6 \times 9 = 54 \quad 73 \times 17 = 1241$ (v)

$\therefore 676 \times 89 = 536 \times 10^2 + (1241 - 536 - 54) \times 10 + 54$

$= 53600 + 6510 + 54$

$= 60164$

x) $98 \times 67 = (9 \times 10^1 + 8)(6 \times 10^1 + 7)$

$9 \times 6 = 54 \quad 8 \times 7 = 56 \quad 17 \times 13 = 221$ (vi)

$\therefore 98 \times 67 = 54 \times 10^2 + (221 - 54 - 56) \times 10 + 56$

$= 5400 + 1110 + 56$

$= 6566$

xi) $774 \times 156 = (77 \times 10^1 + 4)(15 \times 10^1 + 6)$

$77 \times 15 = 1155$ (vii) $\quad 4 \times 6 = 24 \quad 81 \times 21 = 1701$ (viii)

$\therefore 774 \times 156 = 1155 \times 10^2 + (1701 - 1155 - 24) \times 10 + 24$

$= 115500 + 5220 + 24 = 120744$

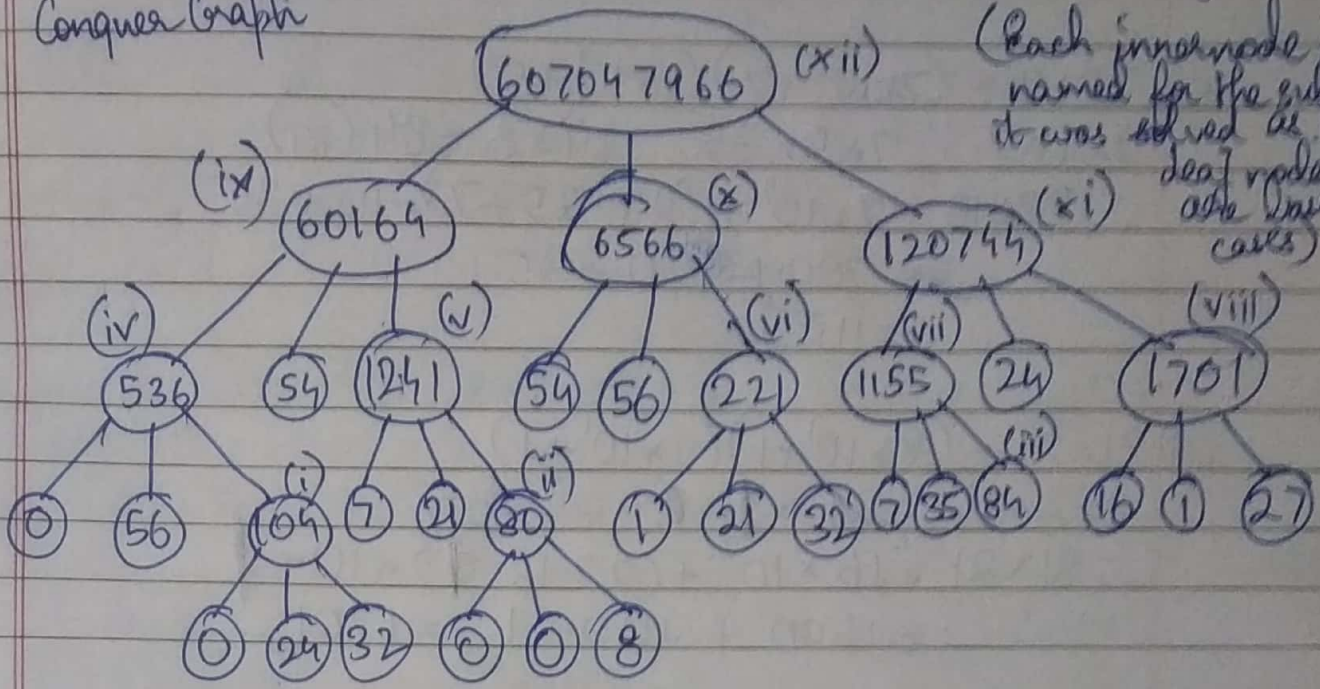
xii) $67698 \times 8967 = (676 \times 10^2 + 98)(89 \times 10^2 + 67)$

$676 \times 89 = 60164$ (ix) $\quad 98 \times 67 = 6566$ (x) $\quad 774 \times 156 = 120744$ (xi)

$\therefore 67698 \times 8967 = 60164 \times 10^4 + (120744 - 60164 - 6566) \times 10^2 + 6566$

$= 601640000 + 5401400 + 6566 = 607047966$

Conquer Graph



(Each inner node is named for the subproblem it was solved as. Leaf nodes are base cases)

mention in divide graph also

5. Professor Caesar wishes to develop a matrix-multiplication algorithm that is asymptotically faster than Strassen's algorithm. His algorithm will use the divide-and-conquer method, dividing each matrix into pieces of size $\frac{n}{4} \times \frac{n}{4}$, and the divide and combine steps together will take $\Theta(n^2)$ time. He needs to determine how many subproblems his algorithm has to create in order to beat Strassen's algorithm. If his algorithm creates a subproblems, then the recurrence for the running time $T(n)$ becomes $T(n) = aT\left(\frac{n}{4}\right) + \Theta(n^2)$. What is the largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than Strassen's algorithm?

His recurrence relation is
$T(n) = aT(n/4) + \Theta(n^2)$

Consider the master theorem.

i) $f(n) = \Theta(n^2)$ $O(n^{\log_4 a})$

$\Theta(n^2) = O(n^{\log_4 a})$

$\therefore n^2 < n^{\log_4 a}$

$2 < \log_4 a$

$a^2 = 4 \Rightarrow a = 2 \Rightarrow a = 4^2 = 16$

$4^2 < \log_4 a$

$a > 16$.

$\Rightarrow T(n) = O(n^{\log_4 a})$.

$T(n) < O(n^{\log_2})$

$\therefore \log a < \log 7$

$\Rightarrow \frac{1}{2}\log_4 a < \log_2 7$

$\Rightarrow \log_2 a < \log_2 49$

$\Rightarrow a < 49 \Rightarrow a = 48$.

ii) $f(n) = \Theta(n^{\log_b a})$

$\Theta(n^2) = \Theta(n^{\log_4 a})$

$\Rightarrow 2 = \log_4 a \Rightarrow a = 16$ // (End proof here in exam, $\because 16 < 48$)

$\therefore T(n) = \Theta(n^{\log_4 16} \log n) = \Theta(n^2 \log n)$

Lemma: $\log n = O(n^c) \ \forall c > 0$.

Proof:

Since $\log x$ is an increasing function,

First, consider the function $\frac{\log x}{cx}$, $x \in \mathbb{R}^+$.

$\frac{d}{dx}\frac{\log x}{cx} = \frac{\frac{1}{x} - c\log x}{cx^2} < 0 \ \forall x > 2$.

$\therefore \log x / cx$ is decreasing $\forall x > 2$.

$\therefore \log x < cx$ for large x.

$\Rightarrow \log x < cx$ for large x $(x > 2/c)$

$\therefore \log x < cx$ for larger x $(\forall x \ x/\log x > 1/c)$

$\therefore \log(\log x) < c\log x$ for large x $(\forall x \ \log x/\log\log x > 1/c)$

$\Rightarrow \log x < x^c$ for large x $(\forall x \ x/\log\log x > 1/c)$

$\Rightarrow \Theta(n^2 \log n) < \Theta(n^{\log_2 2})$ //

iii) $f(n) = \Omega(n^{\log_b a + \epsilon})$ & $af(n/b) \le cf(n) \ \exists c < 1$

$\Theta(n^2) \Rightarrow \Omega(n^2)$

$\therefore \Omega(n^2) > \Omega(n^{\log_4 a + \epsilon})$

$\Rightarrow n^2 > \log_4 a \Rightarrow 16 > a$ // $\therefore$ This case can also be safely ignored.

∴ Max no. of subproblems = 48 // (Prove in the exam using recursion tree if possible)

** Important lemma : $\log n < n^c$ $(\forall c>0 \quad \forall n \quad \log n / \log\log n > 1/c)$

Proof (Repeated for cleanliness):-

Consider $f(x) = \frac{\log x}{cx}$, $x \in \mathbb{R}^+$, $c \in \mathbb{R}^+$

$$f'(x) = \frac{cx \cdot 1/x - \log x \cdot c}{c^2 x^2} = \frac{c - c\log x}{c^2 x^2} = \frac{1 - \log x}{cx^2}$$

$\because c>0$, $f'(x) < 0 \Rightarrow \log x > 1 \Rightarrow x > 2$ (Assuming base 2, base is irrelevant eventually)

∴ $f(x)$ is decreasing for large x.

∴ $\frac{\log x}{cx}$ is decreasing for large x

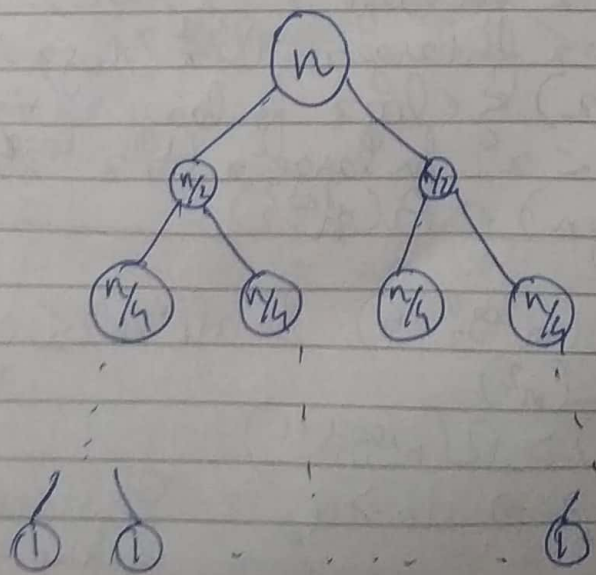
∴ $\log x < cx$ for large x $(\forall x \quad x/\log x > 1/c)$

$\Rightarrow \log(\log x) < c \log x$ for large x $(\forall x \quad \log x / \log(\log x) > 1/c)$

$\Rightarrow \log x < x^c$ for large x $(\forall x \quad \log x / \log\log x > 1/c)$

6. (a) Find the number of nodes in the divide and conquer graph for computing FFT of a vector of length n (for simplicity you can assume n to be a power of 2).
   (b) Now find time complexity of the FFT algorithm by only considering the structure of the divide and conquer graph (without solving any recursion).

a)



Level (mention that level is from top to bottom & describe it)

0

1

2

i

∴ Max no. of subproblems = 48 // (Prove in the exam using recursion tree if possible)

** Important lemma : $\log n < n^c$ $(\forall c > 0)$ $(\forall n \ \log n / \log\log n > 1/c)$

Proof (Repeated for cleanliness):-

Consider $f(x) = \frac{\log x}{cx}$, $x \in \mathbb{R}^+$, $c \in \mathbb{R}^+$

$$f'(x) = \frac{cx \cdot 1/x - \log x \cdot c}{c^2 x^2} = \frac{c - c\log x}{c^2 x^2} = \frac{1 - \log x}{cx^2}$$

$\because c > 0$, $f'(x) < 0 \Rightarrow \log x > 1 \Rightarrow x > 2$ (Assuming base 2, base is irrelevant eventually)

$\therefore f(x)$ is decreasing for large x.

$\therefore \frac{\log x}{cx}$ is decreasing for large x

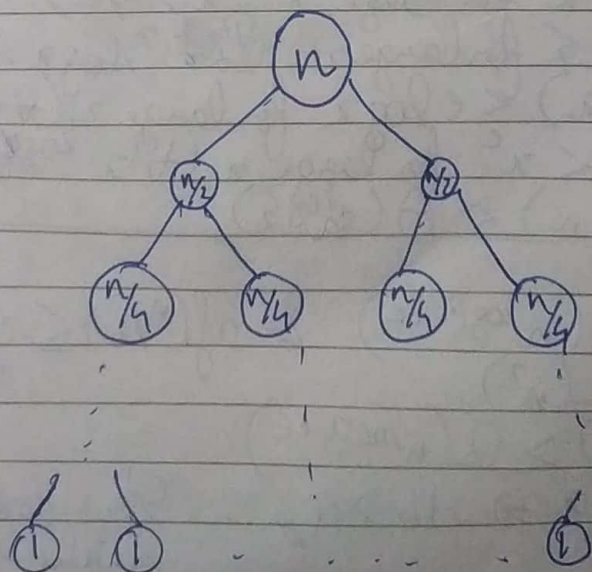
$\therefore \log x < cx$ for large x $(\forall x \ x/\log x > 1/c)$

$\Rightarrow \log(\log x) < c \log x$ for large x $(\forall x \ \log x / \log(\log x) > 1/c)$

$\Rightarrow \log x < x^c$ for large x $(\forall x \ \log x / \log\log x > 1/c)$

6. (a) Find the number of nodes in the divide and conquer graph for computing FFT of a vector of length n (for simplicity you can assume n to be a power of 2).
   (b) Now find time complexity of the FFT algorithm by only considering the structure of the divide and conquer graph (without solving any recursion).

a)



Level (mention that level is from top to bottom & describe it)

0

1

2

k

Scanned with CamScanner

Level	Size of subproblem	No of nodes
0	n	1
1	$n/2$	2
2	$n/4$	4
⋮	⋮	⋮
i	$n/2^i$	2^i
⋮	⋮	⋮
k	1	2^k

$\therefore \frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \lg n$. ($n$ is a power of 2)

$\therefore$ Total no of nodes $= 1 + 2 + 4 + \ldots + 2^k$

$= \frac{1(2^{k+1} - 1)}{2-1} = 2^{k+1} - 1 = 2 \cdot 2^k - 1 = 2n - 1$ //

b) ~~At each~~ ~~Each~~ All non recursive steps at each node with subproblem size $= n/2^i$ take ~~$\Theta(2^i)$~~ $O(n/2^i)$ time.

Thus at each level i, total time taken is
no of nodes × time spent at each node $= 2^i \times O(n/2^i) = O(n)$

Total time for algorithm = time spent at each level × no of levels
$= O(n) \times (k+1) = O(n) \times (\lg n + 1) = O(n \lg n) + O(n)$
$= O(n \lg n)$ //

7. Find 1234 × 4321 using the FFT algorithm showing its divide and conquer graphs.

$1234 = 1 \times 10^3 + 2 \times 10^2 + 3 \times 10^1 + 4 = p(10)$ where $p(x) = x^3 + 2x^2 + 3x + 4$
$4321 = 4 \times 10^3 + 3 \times 10^2 + 2 \times 10^1 + 1 = q(10)$ where $q(x) = 4x^3 + 3x^2 + 2x + 1$

$\therefore 1234 \times 4321 = p(10)\, q(10) = r(10)$, where $r(x) = p(x) q(x)$

The highest power in $s(x)$ will be $x^3 \cdot x^3 = x^6$. This polynomial will have 7 terms at most. To simplify things, take $n = 8$.

$\therefore p(x) = 0x^7 + 0x^6 + 0x^5 + 0x^4 + x^3 + 2x^2 + 3x + 4$

$q(x) = 0x^7 + 0x^6 + 0x^5 + 0x^4 + 4x^3 + 3x^2 + 2x + 1$

$\therefore p = [0, 0, 0, 0, 1, 2, 3, 4] \cdot [4, 3, 2, 1, 0, 0, 0, 0]$

$q = [0, 0, 0, 0, 4, 3, 2, 1] \quad [1, 2, 3, 4, 0, 0, 0, 0]$

Divide and Conquer graph for p

Explain what ω_n is

FFT([4, 3, 2, 1, 0, 0, 0, 0]) = $[4 + 3\omega_8 + 2\omega_8^2 + \omega_8^3, 2 + 2\omega_8, 4 - 3\omega_8 + 2\omega_8^2 - \omega_8^3, 2 - 2\omega_8, 4 - \omega_8 - 2\omega_8^2 - 3\omega_8^3, 2, 4 + 5 + 1 - 10]$

FFT([4, 2, 0, 0]) = $[4 + 2\omega_4, 4 + 2\omega_4^2, 4 - 2\omega_4, 4 - 2\omega_4^2]$

FFT([3, 1, 0, 0]) = $[3 + \omega_4, 3 + \omega_4^2, 3 - \omega_4, 3 - \omega_4^2]$

FFT([4, 0]) = $[4 + 0\omega_2, 4 - 0\omega_2]$

FFT([2, 0]) = $[2 + 0\omega_2, 2 - 0\omega_2]$

FFT([3, 0]) = $[3 + 0\omega_2, 3 - 0\omega_2]$

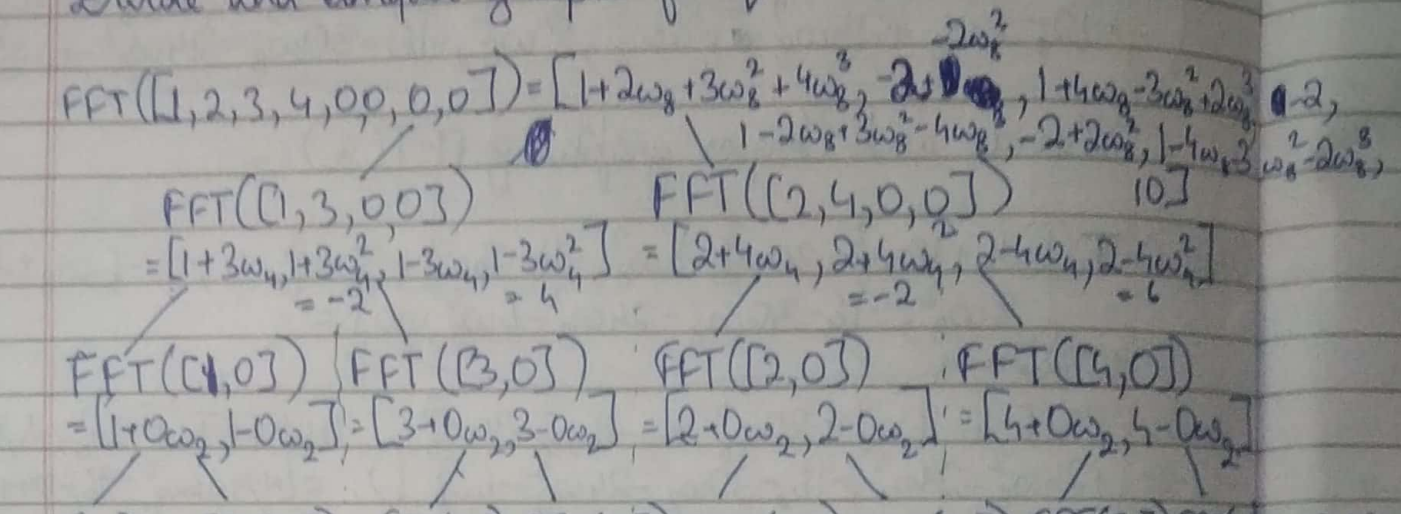
FFT([1, 0]) = $[1 + 0\omega_2, 1 - 0\omega_2]$

FFT([4]) = 4 FFT([0]) = 0 FFT([2]) = 2 FFT([0]) = 0 FFT([3]) = 3 FFT([0]) = 0 FFT([1]) = 1 FFT([0]) = 0

$\therefore FFT(p) = [4 + 3\omega_8 + 2\omega_8^2 + \omega_8^3, 2 + 2\omega_8^2, 4 + \omega_8 - 2\omega_8^2 + 3\omega_8^3, 2, 4 - 3\omega_8 + 2\omega_8^2 + \omega_8^3, 2 - 2\omega_8^2, 4 - \omega_8 - 2\omega_8^2 - 3\omega_8^3, [0]]$

$= -16 - 30\omega_8 - 24\omega_8^2 - 6\omega_8^3$

Divide and Conquer graph for q:



$\therefore r = pq = [(4+3\omega_8+2\omega_8^2+\omega_8^3)(1+2\omega_8+3\omega_8^2+4\omega_8^3), (-2-2\omega_8^2)(2+2\omega_8^2),$
$(1+4\omega_8-3\omega_8^2+2\omega_8^3)(4+\omega_8-2\omega_8^2+3\omega_8^3), (2)(-2),$
$(4-3\omega_8+2\omega_8^2+\omega_8^3)(1-2\omega_8+3\omega_8^2-4\omega_8^3), (-2+2\omega_8^2)(2-2\omega_8^2),$
$(4-\omega_8-2\omega_8^2-3\omega_8^3)(1-4\omega_8-3\omega_8^2-2\omega_8^3), (10)(10)]$

i) $(4+3\omega_8+2\omega_8^2+\omega_8^3)(1+2\omega_8+3\omega_8^2+4\omega_8^3)$
$= 4+8\omega_8+12\omega_8^2+16\omega_8^3+3\omega_8+6\omega_8^2+9\omega_8^3-12+2\omega_8^2+4\omega_8^3-6-8\omega_8$
$+\omega_8^3-2-3\omega_8-4\omega_8^2 = -16+16\omega_8^2+30\omega_8^3$

ii) $(-2-2\omega_8^2)(2+2\omega_8^2) = -(4-4+8\omega_8^2) = -8\omega_8^2$

iii) $(1+4\omega_8-3\omega_8^2+2\omega_8^3)(4+\omega_8-2\omega_8^2+3\omega_8^3) = 4+\omega_8-2\omega_8^2+3\omega_8^3+$
$16\omega_8+4\omega_8^2-8\omega_8^3-12-12\omega_8^2-3\omega_8-6+9\omega_8+8\omega_8^3-2+4\omega_8-6\omega_8$
$= -16+30\omega_8-16\omega_8^2$

iv) $(2)(-2) = -4$

viii) $(10)(10) = 100$

v) $(4-3\omega_8+2\omega_8^2+\omega_8^3)(1-2\omega_8+3\omega_8^2-4\omega_8^3)$
$= 4-8\omega_8+12\omega_8^2-16\omega_8^3-3\omega_8+6\omega_8^2-9\omega_8^3-12+2\omega_8^2-4\omega_8^3-6+8\omega_8$
$+\omega_8^3+2-3\omega_8+4\omega_8^2 = -12-6\omega_8+24\omega_8^2-28\omega_8^3$

vi) $-(2-2\omega_8^2)(2-2\omega_8^2) = -(4-4-8\omega_8^2) = 8\omega_8^2$

vii) $(4-\omega_8-2\omega_8^2-3\omega_8^3)(1-4\omega_8-3\omega_8^2-2\omega_8^3) = 4-16\omega_8-12\omega_8^2-8\omega_8^3$
$-\omega_8-4\omega_8^2-3\omega_8^3-2-2\omega_8^2+8\omega_8^3-6-4\omega_8-3\omega_8^3-12-9\omega_8-6\omega_8^2$ $= -16-30\omega_8$

move to the beginning

$x = [\ : FFT(x) = [-16+16\omega_8^2+30\omega_8^3, -8\omega_8^2, -16-16\omega_8^2+30\omega_8^3, -4,$
$-12-6\omega_8+24\omega_8^2+28\omega_8^3, 8\omega_8, -16-30\omega_8-24\omega_8^2-6\omega_8^3, 100]$

Divide and Conquer graphs of FFT(x):-

$\omega_n = e^{2\pi i/n}$

INV_FFT($[-16+16\omega_8^2+30\omega_8^3, -8\omega_8^2, -16-16\omega_8^2+30\omega_8^3, -4, -12-6\omega_8+24\omega_8^2-28\omega_8^3,$
$8\omega_8^2, -16-30\omega_8-24\omega_8^2-6\omega_8^3, 100]$) =

move to the beginning

INV_FFT($[-16+16\omega_8^2+30\omega_8^3, -16-16\omega_8^2+30\omega_8^3, -12-6\omega_8+24\omega_8^2-28\omega_8^3, -16-30\omega_8-24\omega_8^2-6\omega_8^3]$) =

INV_FFT($[-8\omega_8^2, -4, 8\omega_8^2, 100]$)
$= [104-16\omega_8^2, -96, -104-16\omega_8^2, 96]$

INV_FFT($[-16+16\omega_8^2+30\omega_8^3, -12-6\omega_8+24\omega_8^2-28\omega_8^3]$)
$= [-4+6\omega_8-8\omega_8^2+58\omega_8^3, 28-6\omega_8+40\omega_8^2+24\omega_8^3]$

INV_FFT($[-16-30\omega_8-24\omega_8^2-6\omega_8^3]$) $= [30\omega_8+36\omega_8^2+36\omega_8^3, -32-30\omega_8-40\omega_8^2+24\omega_8^3]$

INV_FFT($[-8\omega_8^2, 8\omega_8^2]$) $= [-16\omega_8^2, 0]$

INV_FFT($[-4, 100]$) $= [-104, 96]$

(Draw in exam)

INV_FFT($[-16+16\omega_8^2+30\omega_8^3]$)
$= [-16+16\omega_8^2+30\omega_8^3]$

INV_FFT($[-12-6\omega_8+24\omega_8^2-28\omega_8^3]$)
$= [-12-6\omega_8+24\omega_8^2-28\omega_8^3]$

(ω_n^{3k})

(ω_n^{3k})

ω_n^{3k} $3k \le n/3$

8. (a) Describe the generalization of the FFT procedure to the case in which n is a power of 3 (using three subproblems). Give a recurrence for the running time, and solve the recurrence.

(b) Find 97×68 using the above algorithm showing its divide and conquer graphs.

Consider a polynomial $a(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1}$
The FFT is used to find its value at $\omega_n, \omega_n^2, \dots, \omega_n^n$; $\omega_n = e^{2\pi i/n}$

Note that $a(x) = a^{[0]}(x^3) + x a^{[1]}(x^3) + x^2 a^{[2]}(x^3)$
where $a^{[0]}(x)$ = polynomials with coefficients of $a(x)$ indexed as elements of equivalence class of 0 under $\equiv \pmod 3$.
and $a^{[1]}(x)$ and $a^{[2]}(x)$ are defined similarly

Now consider $0 < k \le n/3$ for some k with
$a(\omega_n^k) = a^{[0]}(\omega_n^{3k}) + \omega_n^k a^{[1]}(\omega_n^{3k}) + \omega_n^{2k} a^{[2]}(\omega_n^{3k})$

But $a(\omega_n^{k+n/3}) = a^{[0]}(\omega_n^{3k+n}) + \omega_n^{k+n/3} a^{[1]}(\omega_n^{3k+n}) + \omega_n^{2k+2n/3} a^{[2]}(\omega_n^{3k+n})$
and $\omega_n^{3k+n} = \omega_n^n \omega_n^{3k} = \omega_n^{3k}$
$\omega_n^{k+n/3} = \omega_n^k \cdot \omega_n^{n/3} = \omega_n^k \left(e^{\frac{2\pi i}{n} \cdot \frac{n}{3}}\right) = \omega_n^k \left(e^{\frac{2\pi i}{3}}\right) = \omega_n^k \omega_3$
$\omega_n^{2k+2n/3} = \omega_n^{2k} \cdot \omega_n^{2n/3} = \omega_n^{2k} \cdot \omega_3^2$

$\therefore a(\omega_n^{k+n/3}) = a^{[0]}(\omega_n^{3k}) + \omega_3 \omega_n^k a^{[1]}(\omega_n^{3k}) + \omega_3^2 \omega_n^{2k} a^{[2]}(\omega_n^{3k})$
and $a(\omega_n^{k+2n/3}) = a^{[0]}(\omega_n^{3k}) + \omega_3^2 \omega_n^k a^{[1]}(\omega_n^{3k}) + \omega_3 \omega_n^{2k} a^{[2]}(\omega_n^{3k})$

$\omega_3 = e^{2\pi i/3} = \cos 2\pi/3 + i \sin 2\pi/3 = -1/2 + \sqrt{3}i/2$
$\omega_3^2 = e^{4\pi/3} = \cos 4\pi/3 + i \sin 4\pi/3 = -1/2 - \sqrt{3}i/2$

Thus we only need to find $a^{[0]}(\omega_n^{3k})$, $a^{[1]}(\omega_n^{3k})$ and $a^{[2]}(\omega_n^{3k})$ recursively to find $a(\omega_n^k)$, $0 < k \le n$.

Finally, we have $\omega_n^{3k} = e^{2\pi i/n \cdot 3k} = \left(e^{2\pi i/n/3}\right)^k = \omega_{n/3}^k$
$\therefore$ We have our recursion fully established.

a) Full algorithm:

R-FFT-3(a)

1) n = length(a)
2) if ($n == 1$) return a.
3) $\omega = 1$, $\omega_3 = 1$
4) $\omega_n = e^{2\pi i/n}$
5) $\omega_3 = e^{2\pi i/3}$
6) $y^{[0]} = a^{[0]}$
7) $y^{[0]} = $ R-FFT-3$(y^{[0]})$
8) $y^{[1]} = a^{[1]}$
9) $y^{[1]} = $ R-FFT-3$(y^{[1]})$
10) $y^{[2]} = a^{[2]}$
11) $y^{[2]} = $ R-FFT-3$(y^{[2]})$
12) for $k =$ to $n/3$
    - 12.1) $\omega = \omega\omega_n$
    - 12.2) $\omega_3 = \omega_3\omega_n$
    - 12.3) $\omega_3 = \omega_3\omega_n$
    - 12.4) $y[k] = y^{[0]}[k] + \omega y^{[1]}[k] + \omega_3 y^{[2]}[k]$
    - 12.5) $y[k+n/3] = y^{[0]}[k] + \omega_3\omega y^{[1]}[k] + \omega_3^2\omega_3 y^{[2]}[k]$
    - 12.6) $y[k+2n/3] = y^{[0]}[k] + \omega_3^2\omega y^{[1]}[k] + \omega_3\omega_3 y^{[2]}[k]$
13) return y.

Absolutely crucial — interchange 12.4 with 12.6

$$T(n) = 3T(n/3) + \Theta(n).$$

$a = b = 3$

$$\Theta(n^{\log_b a}) = \Theta(n^{\log_3 3}) = \Theta(n) //$$

$$\therefore T(n) = \Theta(n^{\log_3 3} \lg n) = \Theta(n \lg n) //$$

$+332\omega_3^2$]

a) Full algorithm:

R-FFT-3(a)

1) n = length(a)
2) if ($n == 1$) return a.
3) $\omega = 1$, $\omega_s = 1$
4) $\omega_n = e^{2\pi i/n}$
5) $\omega_3 = e^{2\pi i/3}$
6) $y^{[0]} = a^{[0]}$
7) $y^{[0]} = $ R-FFT-3$(y^{[0]})$
8) $y^{[1]} = a^{[1]}$
9) $y^{[1]} = $ R-FFT-3$(y^{[1]})$
10) $y^{[2]} = a^{[2]}$
11) $y^{[2]} = $ R-FFT-3$(y^{[2]})$
12) for $k =$ to $n/3$
    - 12.1) $\omega = \omega\omega_n$
    - 12.2) $\omega_s = \omega_s\omega_n$
    - 12.3) $\omega_s = \omega_s\omega_n$
    - 12.4) $y[k] = y^{[0]}[k] + \omega y^{[1]}[k] + \omega_s y^{[2]}[k]$
    - 12.5) $y[k+n/3] = y^{[0]}[k] + \omega_3\omega y^{[1]}[k] + \omega_3^2\omega_s y^{[2]}[k]$
    - 12.6) $y[k+2n/3] = y^{[0]}[k] + \omega_3^2\omega y^{[1]}[k] + \omega_3\omega_s y^{[2]}[k]$
13) return y.

interchange 12.5 with 12.6 — absolutely crucial

$$T(n) = 3T(n/3) + \Theta(n).$$

$a = b = 3$

$$\Theta(n^{\log_b a}) = \Theta(n^{\log_3 3}) = \Theta(n) //$$

$$\therefore T(n) = \Theta(n^{\log_3 3} \lg n) = \Theta(n \lg n) //$$

$3 3 2 \omega_3^2$

b) $97 = 7 + 9\times10 + 0\times10^2 = p(10)$, $p(x) = 0x^2 + 9x + 7$

$68 = 8 + 6\times10 + 0\times10^2 = q(10)$, $q(x) = 0x^2 + 6x + 8$

$\therefore 97\times68 = r(10)$, $r(x) = p(x)q(x)$

Divide and Conquer Graph for p

move to beginning

$FFT([7,9,0]) = [7+9\omega_3, 7+9\omega_3^2, 16]$

$FFT([7]) = [7]$ $FFT([9]) = [9]$ $FFT([0]) = [0]$

Divide and Conquer Graph for q

move to beginning

$FFT([8,6,0]) = [8+6\omega_3, 8+6\omega_3^2, 14]$

$FFT([8]) = [8]$ $FFT([6]) = [6]$ $FFT([0]) = [0]$

move to beginning

$r = pq = [56+114\omega_3+54\omega_3^2, 56+114\omega_3^2+54\omega_3, 224]$

$= [2+60\omega_3, 2+60\omega_3^2, 224]$

move to beginning

Divide and Conquer Graph for FFT(r):

$INV\text{-}FFT([56+114\omega_3+54\omega_3^2, 56+114\omega_3^2+54\omega_3, 224])$

$= [56+114\omega_3+54\omega_3^2 + 56\omega_3^2 + 114\omega_3 + 54 + 224\omega_3,$

$56+114\omega_3+54\omega_3^2+56+114\omega_3^2+54\omega_3+224,$

$56+114\omega_3+54\omega_3^2+56\omega_3+114+54\omega_3^2+224\omega_3^2]$

$= [110+452\omega_3+110\omega_3^2, 336+168\omega_3+168\omega_3^2, 170+170\omega_3+332\omega_3^2]$

$= [342\omega_3, 168,$

INV-FFT([2+60ω_3, 2+60ω_3^2])

INV-FFT([224, 2+60ω_3, 2+60ω_3^2])

$= [224+2+60\omega_3+2+60\omega_3^2,\ 224+2\omega_3^2+60+2\omega_3+60,$
$224+2\omega_3+60\omega_3^2+2\omega_3^2+60\omega_3] = [168, 342, 162]$

INV-FFT([224]) = [224]

INV-FFT([2+60ω_3]) = [2+60ω_3]

INV-FFT([2+60ω_3^2]) = [2+60ω_3^2]

$\therefore r = \frac{1}{\text{len}(r)}[168, 342, 162] = \frac{1}{3}[168, 342, 162] = [56, 114, 54]$

$\therefore r(x) = 56 + 114x + 54x^2 \Rightarrow r(10) = 56 + 1140 + 5400 = 6596$

$\therefore 97 \times 68 = 6596$

9. Consider an $n \times n$ grid graph G. (An $n \times n$ grid graph is just the adjacency graph of an $n \times n$ chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of natural numbers (i, j), where $1 \leq i \leq n$ and $1 \leq j \leq n$; the nodes (i, j) and (k, l) are joined by an edge if and only if $|i - k| + |j - l| = 1$.) Each node v of G is labeled with a real number x_v. You may assume that the real numbers labeling the nodes are all distinct. A node v of G is a *local minimum* if the label x_y is less than the label x_w for all nodes w that are joined to v by an edge. You are given such an $n \times n$ grid graph G, but the labeling is only specified in the following *implicit* way: for each node v, you can determine the value x_v by probing the node v. Show how to find a local minimum of G using only $O(n)$ *probes* to the nodes of G. Give a proof of correctness of your algorithm and also prove its time complexity.

$x_v \leq x_w$??

The logic here is simple divide and conquer. Given that the element is in the grid, the element must be in some quadrant of the grid. Thus, the task at hand requires searching through quadrants recursively after finding any such quadrant. The finding of the quadrant requires some ingenuity.

Firstly, for an $n \times n$ grid, define the four quadrants by row $R_{n/2}$ and column $C_{n/2}$. Probe all the $2n+1$ nodes in $R_{n/2} \cup C_{n/2}$ and find the node v with minimum x_v. Assume w.l.o.g. of generality that v is in $C_{n/2}$.

Probe the two nodes w and x on the left and right of v respectively. If, now $x_v < x_w$ and $x_v < x_y$ then v is a local minimum of the grid. On the other hand, if, w/o loss of generality $x_w < x_v$, then x_w is lesser in value than all elements in the row and column $R_{n/2}$ & $C_{n/2}$ respectively that border the quadrant containing it. Then recursively probe this quadrant.

Why is this algorithm correct? If x_w is the smallest element in its quadrant, it is already the local maximum. If not it is not even a local maximum, it must have an adjacent element smaller than itself. But this is still smaller than $R_{n/2}$ & $C_{n/2}$. More formally,

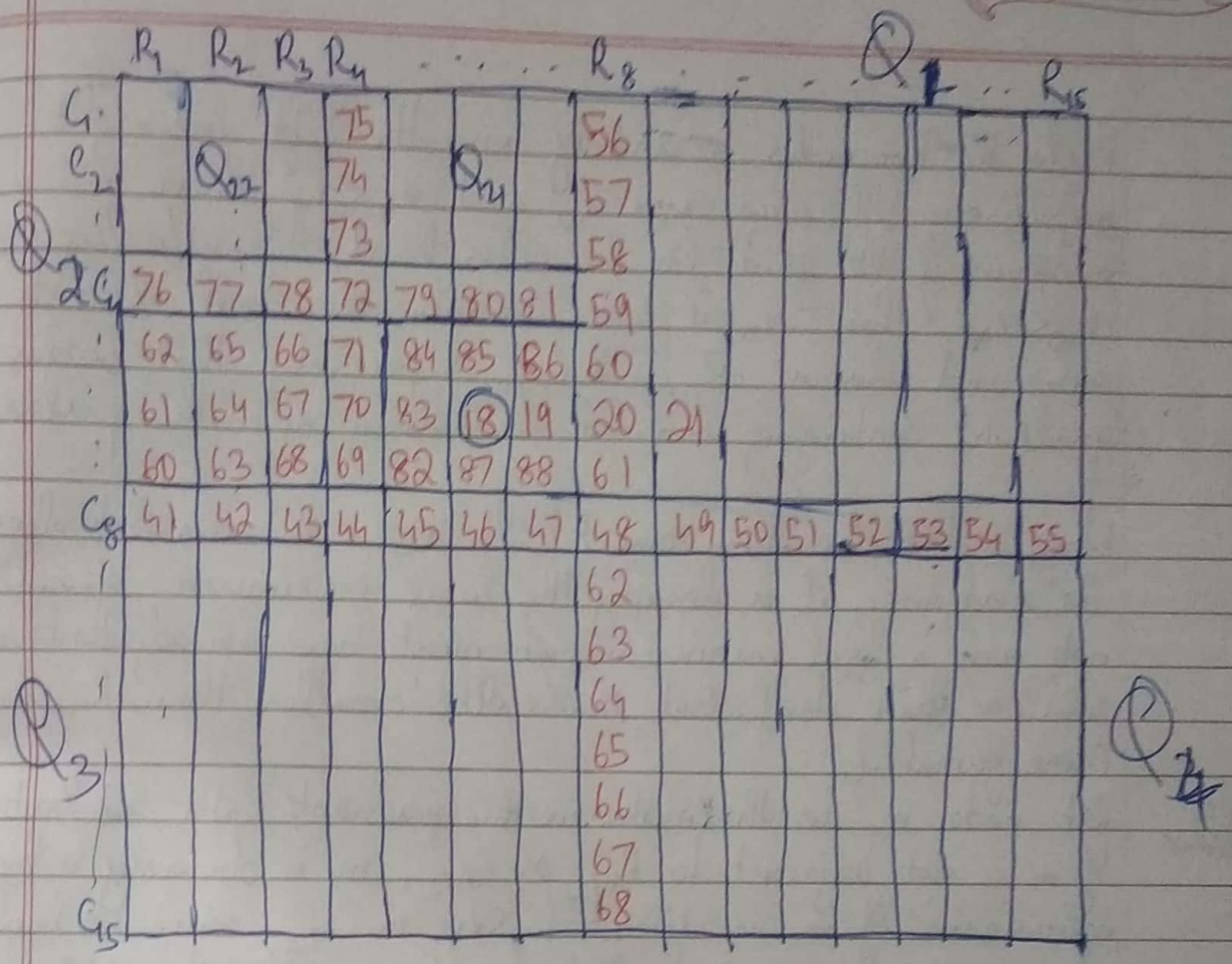
Proof of correctness: Let node m be the node in this quadrant with the least x_m. If m is not adjacent to $R_{n/2}$ or $C_{n/2}$, m is trivially a local minimum. If m is adjacent to $R_{n/2}$ or $C_{n/2}$, since we know $x_m \leq x_w$ and $x_w < x_v$, and $x_v \leq x_z$, $\forall z \in R_{n/2} \cup C_{n/2}$,

$\therefore x_m < x_z \quad \forall z \in R_{n/2} \cup C_{n/2}$.

Thus, x_m is lower in value than any node it is adjacent to on either $R_{n/2}$ or $C_{n/2}$. Since it is the least element in the quadrant also, it is definitely smaller than all elements that surround it. Thus, x_m is a local minimum.

End of proof.

We have thus shown that our recursive search guarantees the existence of a local minimum. There is a small issue here, that is x_w. We rely on $x_m \leq x_w$ This need not be the case immediately. Consider the grid in the next page. $x_y = 20$ and $x_w = 19$. Thus we must recursively probe Q_2. The smallest element in $R_8 \cup C_8$ is now 69. This very obviously leads to issues, as now every element in Q_{23} is larger than every element in C_8, and we now have no local minimum in Q_{23}.



The solution is to notice that every element in $R_4 \cup C_4$ is greater than z_w. Therefore the quadrant Q_{24} is completely bordered by elements larger than z_w, and this is the quadrant we should continue searching in.

localMinimum (grid, rb, re, cb, ce):

1) if (rb == re AND cb == ce)
   1.1) return grid[rb]
2) (maximum, index) $\stackrel{max}{=}$ (max (grid[rb + $\frac{re-rb}{2}$]), max (grid[cb + $\frac{ce-cb}{2}$]))
3) newmin = min (grid[index − 1], grid[index], grid[index + 1])
4) return LocalMinimum (grid, appQuadrant (rb, re, cb, ce, newmin))

↳ Expand appropriately in the e-ram.

10. In the *Josephus Problem*, we start with n people numbered 1 to n around a circle, and we eliminate every *second* remaining person until only one survives. For example, the elimination order for $n = 10$ is 2, 4, 6, 8, 10, 3, 7, 1, 9, so 5 survives. The problem is to determine the survivor's number, $J(n)$ (in the above example, we have $J(10) = 5$). Design a linear time complexity algorithm for computing $J(n)$.

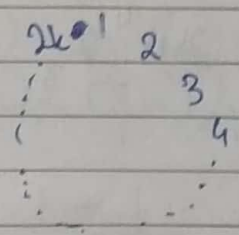
Let's look at a case by case analysis.

i) For $n=1$, survivor number $=1$.

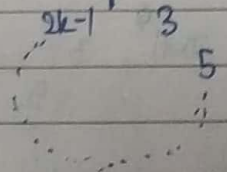
ii) For $n=2$, survivor number $=1$.

iii) For $n=3$, the second person is removed initially. After this, there are two people left, but removal starts from person 1. So, we need to add/subtract 1 from the result for 2, and then if it is the result, return $2(1)-1=3$.

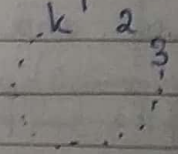
iv) For $n=2k$, if people are seated as



Then, after the first round of elimination, people left are

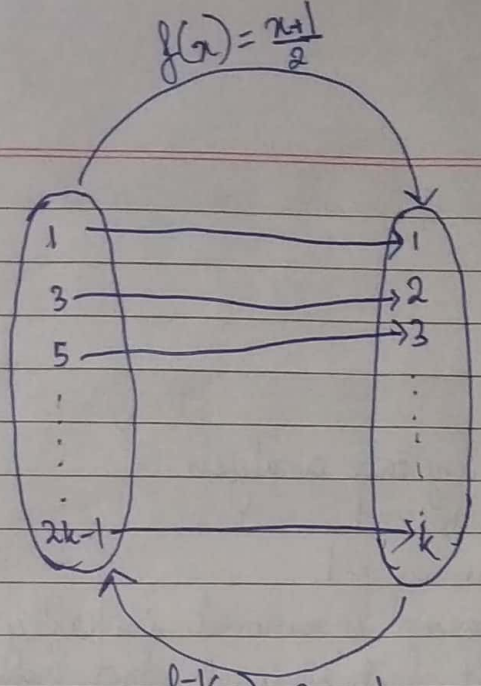


There are thus k people left. To get a one to one correspondence with

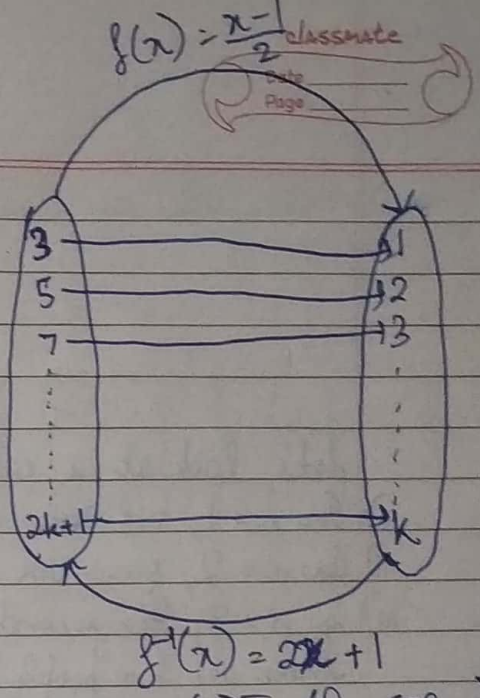


use the following map in (i):

1)
2)
3)
4)



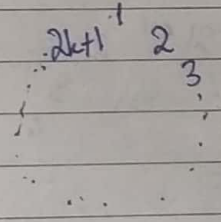
(i) Josephus(2k)



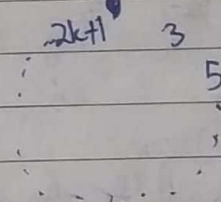
(ii) Josephus(2k+1)

Thus we get Josephus(2k) = Josephus(k) * 2 − 1.

v) In $n = 2k+1$ if people are seated as



After the first round of elimination, the people still left are



(1 has also been removed)

Since people seated at 2(1), 2(2), ..., 2(k−1), 2k and 1 were removed, k+1 people were removed, and k people are left. ~~As we did before, we use~~ ~~Since elimination now starts from 1, we need to add 1 to the result of the subproblem~~ Using the map in (ii), we have

Josephus(2k+1) = Josephus(k) * 2 + 1.

Josephus(x):

1) if x == 1 OR x == 2
   1.1) return 1
2) if x % 2 == 0
   2.1) return Josephus(x/2) * 2 − 1
3) else
   3.1) return Josephus(⌊x/2⌋) * 2 + 1.

Our complexity relation is:

$T(n) = T(n/2) + \Theta(1)$.

$a = 1, b = 2$.

$n^{\log_b^a} = n^{\log_2^1} = n^0 = 1$

→ Master theorem

$\therefore f(n) = \Theta(n^{\log_b^a}) \Rightarrow T(n) = \Theta(1 \lg n) = \Theta(\lg n) = O(n)$

$(\because \Theta(\lg n) \subset O(\lg n) \subset O(n))$

As a demo,

$\{J_n\} = 1, 1, 3, 1, 3, 5, 7, 1, 3, 5 \ldots\ldots\ldots$

Tutorial 3, Design and Analysis of Algorithms, 2020

classmate

1. Solve the following instance of the *Fractional Knapsack Problem*, by applying the *Greedy Algorithm*:
There are 7 items with profit of the items given by $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$, weight of the items given by $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$, and the knapsack capacity given by $W = 15$.

profit/weight array = $(5, 5/3, 3, 1, 6, 9/2, 3)$
Descending sort of p/w array = $(6, 5, 9/2, 3, 3, 5/3, 1)$.
$W = 15$, profit = 0, $= (5, 1, 6, 3, 7, 2, 4)$ Object sort.

i) Object 5 is chosen, $w_5 = 1 < W$
$W = W - 1 = 14$
profit = 6 Knapsack = $\{5\}$

2) Object 1 is chosen, $w_1 = 2 < W$
$W = W - 2 = 12$
profit = $6 + 10 = 16$ Knapsack = $\{5, 1\}$

3) Object 6 is chosen. $w_6 = 4 < W$
$W = W - 4 = 8$
profit = $16 + 18 = 34$ Knapsack = $\{5, 1, 6\}$.

4) Object 3 is chosen. $w_3 = 5 < W$
$W = W - 5 = 3$
profit = $34 + 15 = 49$ Knapsack = $\{5, 1, 6, 3\}$

5) Object 7 is chosen. $w_7 = 1 < W$
$W = W - 1 = 2$
profit = $49 + 3 = 52$ Knapsack = $\{5, 1, 6, 3, 7\}$

6) Object 2 is chosen. $w_2 = 3 > W$.
$W = W - W = 0$
profit = $52 + 2(5/3) = 166/3$ Knapsack = $\{5, 1, 6, 3, 7, 2(2/3)\}$

Since W is now 0, the knapsack is full.
Knapsack = $\{5, 1, 6, 3, 7, 2(2/3)\}$

Tutorial 3, Design and Analysis of Algorithms, 2020

classmate

1. Solve the following instance of the *Fractional Knapsack Problem*, by applying the *Greedy Algorithm*:
There are 7 items with profit of the items given by $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$, weight of the items given by $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$, and the knapsack capacity given by $W = 15$.

profit/weight array = $(5, 5/3, 3, 1, 6, 9/2, 3)$
Decending sort of p/w array = $(6, 5, 9/2, 3, 3, 5/3, 1)$.
$W = 15$, profit = 0, $= (5, 1, 6, 3, 7, 2, 4)$ Object sort.

1) Object 5 is chosen, $w_5 = 1 < W$
$W = W - 1 = 14$
profit = 6 Knapsack = $\{5\}$

2) Object 1 is chosen, $w_1 = 2 < W$
$W = W - 2 = 12$
profit = $6 + 10 = 16$ Knapsack = $\{5, 1\}$

3) Object 6 is chosen. $w_6 = 4 < W$
$W = W - 4 = 8$
profit = $16 + 18 = 34$ Knapsack = $\{5, 1, 6\}$.

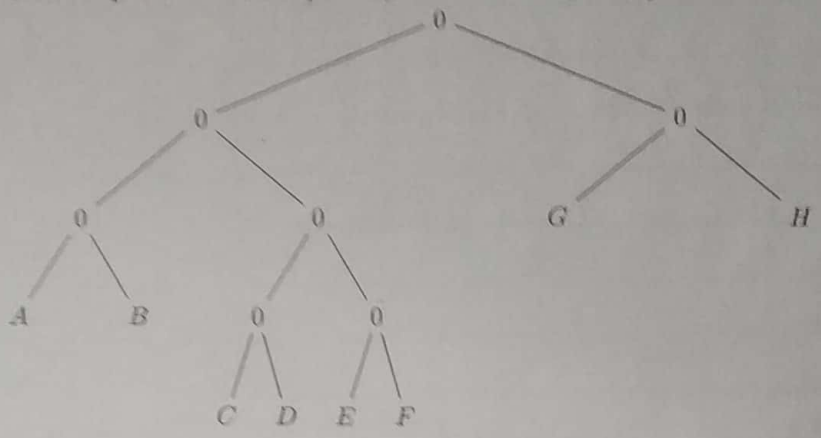
4) Object 3 is chosen. $w_3 = 5 < W$
$W = W - 5 = 3$
profit = $34 + 15 = 49$ Knapsack = $\{5, 1, 6, 3\}$

5) Object 7 is chosen. $w_7 = 1 < W$
$W = W - 1 = 2$
profit = $49 + 3 = 52$ Knapsack = $\{5, 1, 6, 3, 7\}$

6) Object 2 is chosen. $w_2 = 3 > W$.
$W = W - W = 0$
profit = $52 + 2(5/3) = 166/3$ Knapsack = $\{5, 1, 6, 3, 7, 2(2/3)\}$

Since W is now 0, the knapsack is full.
Knapsack = $\{5, 1, 6, 3, 7, 2(2/3)\}$

2. (a) Find the prefix code corresponding to the following binary tree:



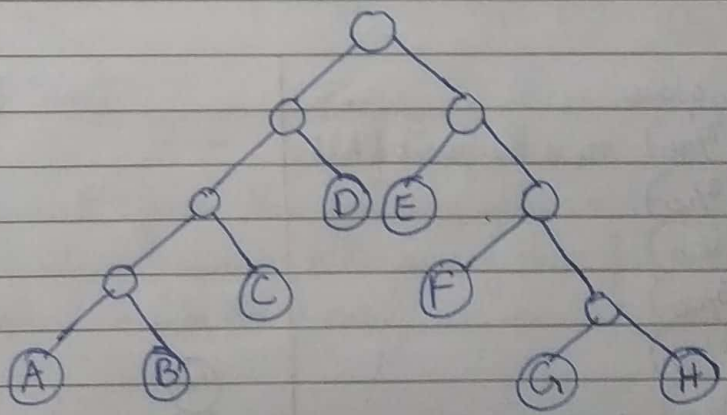
(b) Draw the binary tree corresponding to the following prefix code:
$A = 0000, B = 0001, C = 001, D = 01, E = 10, F = 110, G = 1110, H = 1111.$

(c) Using Huffman's algorithm find the optimal prefix code for the alphabet $\{A, B, C, D, E, F, G, H\}$ for the following frequencies:
$f_A = \frac{1}{40}, f_B = \frac{4}{40}, f_C = \frac{3}{40}, f_D = \frac{10}{40}, f_E = \frac{2}{40}, f_F = \frac{5}{40}, f_G = \frac{6}{40}, f_H = \frac{9}{40}$.

(d) Find *Average Bit Length* of the optimal prefix code in 2(c).

a) A = 000 B = 001 C = 0100 D = 0101
E = 0110 F = 0111 G = 10 H = 11

b)



c) Create a structure to store a node of a tree. Apart from any auxiliary data members required to construct the tree, the node should have a label to identify itself, and a frequency datamember that is the frequency of the letter the node represents, if it is a leaf node. Its frequency is the sum of those of the node's children if it is an internal node.

Create nodes for each character and add all these nodes into a min. priority queue, indexed by the frequency. (implemented as a min heap)

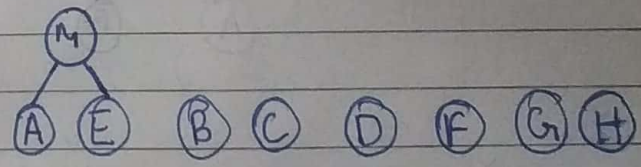
Min priority queue before loop begins:
$(A, f_A = 1/40)$
$(E, f_E = 2/40)$
$(C, f_C = 3/40)$
$(B, f_B = 4/40)$
$(F, f_F = 5/40)$
$(G, f_G = 6/40)$
$(H, f_H = 9/40)$
$(D, f_D = 10/40)$

---

i) Take two elements from the priority queue : $(A, f_A), (E, f_E)$
Create a parent, whose frequency is the sum of those of children : $(n_1, f_{n_1} = f_A + f_E = 3/40)$
Push this node back into the queue.

Priority queue:
$(n_1, f_{n_1} = 3/40)$ n_1 is the parent of A & E.
$(C, f_C = 3/40)$
$(B, f_B = 4/40)$
$(F, f_F = 5/40)$
$(G, f_G = 6/40)$
$(H, f_H = 9/40)$
$(D, f_D = 10/40)$

Tree:



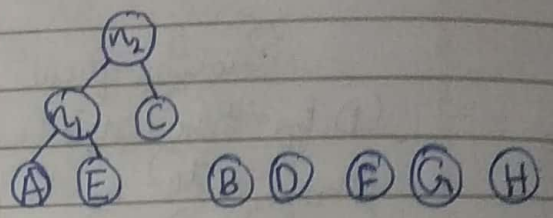
---

ii) Take two elements from the priority queue : (n_1, f_{n_1}), (C, f_C)
Create a parent, whose frequency is the sum of those of children : $(n_2, f_{n_2} = f_{n_1} + f_C = 6/40)$
Push this node back into the queue.

Priority queue

$(B, f_B = 4/40)$ n_1 is the parent of A & E
$(F, f_F = 5/40)$ n_2 is the parent of n_1 & C
$(n_2, f_{n_2} = 6/40)$
$(G_1, f_{G_1} = 6/40)$
$(H, f_H = 9/40)$
$(D, f_D = 10/40)$

Tree:

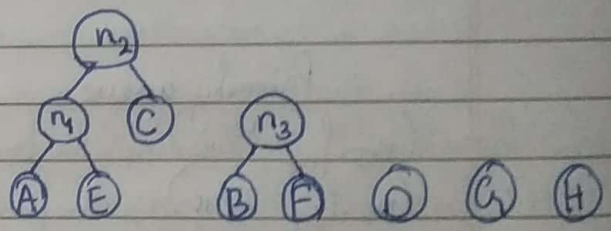


iii) Take two elements from the priority queue: $(B, f_B), (F, f_F)$
Create a parent node whose frequency is the sum of those of children: $(n_3, f_{n_3} = f_B + f_F = 9/40)$
Push this node back into the queue.

Priority queue

$(n_2, f_{n_2} = 6/40)$ n_1 is parent of A & E
$(G_1, f_{G_1} = 6/40)$ n_2 is the parent of n_1 & C
$(n_3, f_{n_3} = 9/40)$ n_3 is the parent of B & F
$(H, f_H = 9/40)$
$(D, f_D = 10/40)$

Tree

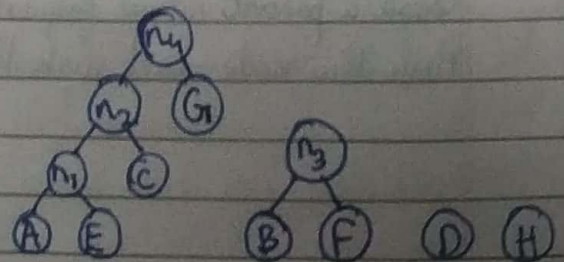


iv) Take two elements from the priority queue: $(n_2, f_{n_2}), (G_1, f_{G_1})$
Create a parent whose frequency is the sum of those of its children: $(n_4, f_4 = f_{n_2} + f_{G_1} = 12/40)$
Push this node back into the queue.

Priority Queue

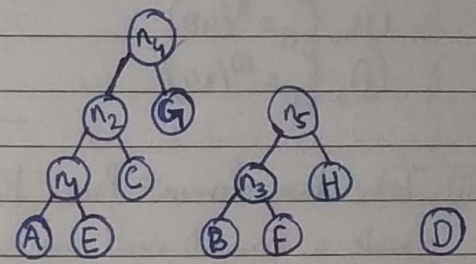
$(n_3, f_{n_3} = 9/40)$ n_4 is the parent of A & E
$(H, f_H = 9/40)$ n_2 is the parent of n_1 & C
$(D, f_D = 10/40)$ n_3 is the parent of B & F
$(n_4, f_{n_4} = 12/40)$ n_4 is the parent of n_2 & G_1

Tree



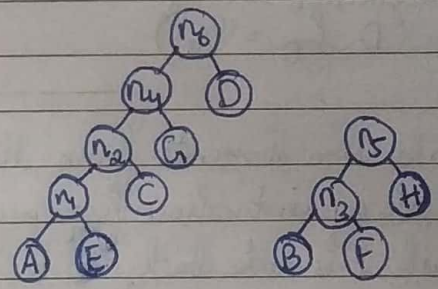
v) Take two elements from the priority queue: (n_3, f_{n_3}), (H, f_H) (is the least)
Create a parent whose frequency is the sum of those of its children: $(n_5, f_{n_5} = f_{n_3} + f_H = 18/40)$
Push this node back into the queue.

Priority queue	Tree
$(D, f_D = 10/40)$	n_4 is the parent of A & E
$(n_4, f_{n_4} = 12/40)$	n_2 is the parent of n_1 & C
$(n_5, f_{n_5} = 18/40)$	n_3 is the parent of B & F
	n_1 is the parent of n_2 & G
	n_5 is the parent of n_3 & H



vi) Take two elements from the priority queue: (D, f_D), (n_4, f_{n_4})
Create a parent whose frequency is the sum of those of its children: $(n_6, f_{n_6} = f_D + f_{n_4} = 22/40)$
Push this node back into the queue.

Priority queue	Tree
$(n_5, f_{n_5} = 18/40)$	n_4 is the parent of A & E
$(n_6, f_{n_6} = 22/40)$	n_2 is the parent of n_1 & C
	n_3 is the parent of B & F
	n_1 is the parent of n_2 & G
	n_5 is the parent of n_3 & H
	n_6 is the parent of D & n_4



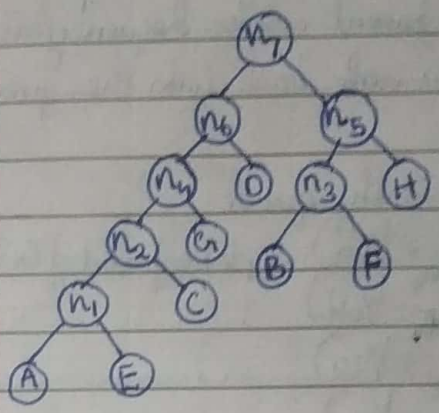
vii) Take two elements from the priority queue: (n_5, f_{n_5}), (n_6, f_{n_6})
Create a parent whose frequency is the sum of those of its children: $(n_7, f_{n_7} = f_{n_5} + f_{n_6} = 40/40)$
Push this node back into the queue.

v)

Priority queue

(n_7, f_{n_7} = 40/40)
n_1 is the parent of A & E
n_2 is the parent of n_1 & C
n_3 is the parent of B & F
n_4 is the parent of n_2 & G
n_5 is the parent of n_3 & H
n_6 is the parent of n_4 & D
n_7 is the parent of n_5 & n_6

Tree



vi) Since the end of the loop has been reached (8 characters, 7 iterations), n_7 is returned and the tree is as above. The prefix code is as follows:

A = 00000 B = 100 C = 0001 D = 01
E = 00001 F = 101 G = 001 H = 11

d) Average Bit length of code = (Bitlength(A) + Bitlength(B) + Bitlength(C) + Bitlength(D) + Bitlength(E) + Bitlength(F) + Bitlength(G) + Bitlength(H))//8
= (5+3+4+2+5+3+3+2)/8 = 27/8 = 3.375//

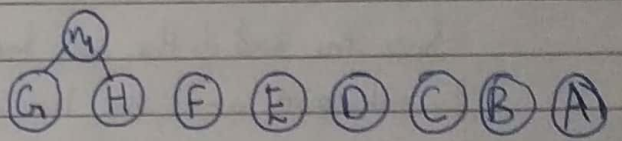
3. Using Huffman's algorithm find the optimal prefix code and *Average Bit Length* of the optimal prefix code for the alphabet $\{A, B, C, D, E, F, G, H\}$ for the following frequencies:
$f_A = \frac{1}{2}, f_B = \frac{1}{4}, f_C = \frac{1}{8}, f_D = \frac{1}{16}, f_E = \frac{1}{32}, f_F = \frac{1}{64}, f_G = \frac{1}{128}, f_H = \frac{1}{128}$.

vii) Write the first two paragraphs as in (2)(c)
No of iterations = no. of nodes − 1 = 8 − 1 = 7//
Min priority queue before the iterations begin:

(H, f_H = 1/128) → (D, f_D = 1/16)
(G, f_G = 1/128) (C, f_C = 1/8)
(F, f_F = 1/64) (B, f_B = 1/4)
(E, f_E = 1/32) (A, f_A = 1/2)

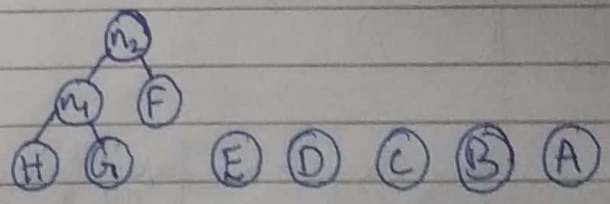
i) Take two elements from the priority queue: $(H, f_H), (G, f_G)$
Create a parent whose frequency is the sum of those of its children: $(n_1, f_{n_1} = f_H + f_G = 1/64)$
Push this node back into the queue.

Priority queue	Tree
$(n_1, f_{n_1} = 1/64)$ n_1 is the parent of G & H	
$(F, f_F = 1/64)$	
$(E, f_E = 1/32)$	
$(D, f_D = 1/16)$	
$(C, f_C = 1/8)$	
$(B, f_B = 1/4)$	
$(A, f_A = 1/2)$	



ii) Take two elements from the priority queue: $(n_1, f_{n_1}), (F, f_F)$
Create a parent whose frequency is the sum of those of its children: $(n_2, f_{n_2} = f_{n_1} + f_F = 1/32)$
Push this node back into the queue.

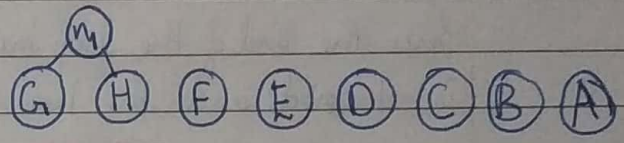
Priority queue	Tree
(n_2, f_{n_2}) n_1 is the parent of G & H	
(E, f_E) n_2 is the parent of n_1 & F	
(D, f_D)	
(C, f_C)	
(B, f_B)	
(A, f_A)	



iii) Take two elements from the priority queue: $(n_2, f_{n_2}), (E, f_E)$
Create a parent whose frequency is the sum of those of its children: $(n_3, f_{n_3} = f_{n_2} + f_E = 1/16)$
Push this node back into the queue.

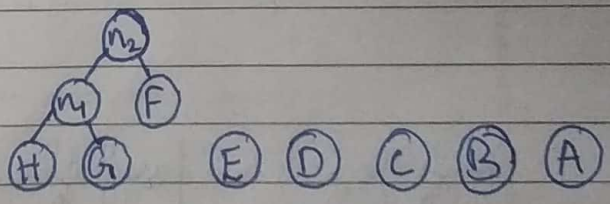
i) Take two elements from the priority queue: (H, f_H), (G, f_G)
Create a parent whose frequency is the sum of those of its children: $(n_1, f_{n_1} = f_H + f_G = 1/64)$
Push this node back into the queue.

Priority queue	Tree
$(n_1, f_{n_1} = 1/64)$ n_1 is the parent of G & H	
$(G, f_G = 1/64)$	
$(E, f_E = 1/32)$	
$(D, f_D = 1/16)$	
$(C, f_C = 1/8)$	
$(B, f_B = 1/4)$	
$(A, f_A = 1/2)$	



ii) Take two elements from the priority queue: (n_1, f_{n_1}), (F, f_F)
Create a parent whose frequency is the sum of those of its children: $(n_2, f_{n_2} = f_{n_1} + f_F = 1/32)$
Push this node back into the queue.

Priority queue	Tree
(n_2, f_{n_2}) n_1 is the parent of G & H	
(E, f_E) n_2 is the parent of n_1 & F	
(D, f_D)	
(C, f_C)	
(B, f_B)	
(A, f_A)	

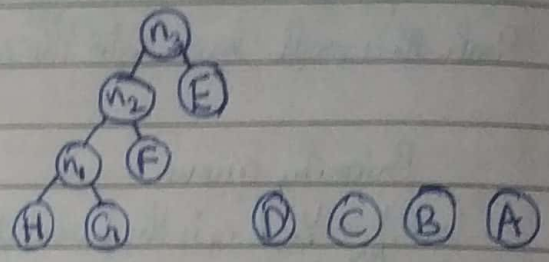


iii) Take two elements from the priority queue: (n_2, f_{n_2}), (E, f_E)
Create a parent whose frequency is the sum of those of its children: $(n_3, f_{n_3} = f_{n_2} + f_E = 1/16)$
Push this node back into the queue.

Priority queue

$(n_3, f_{n_3} = 1/16)$ n_4 is the parent of G & H
$(D, f_D = 1/16)$ n_2 is the parent of n_1 & F
$(C, f_C = 1/8)$ n_3 is the parent of n_2 & E
$(B, f_B = 1/4)$
$(A, f_A = 1/2)$

Tree

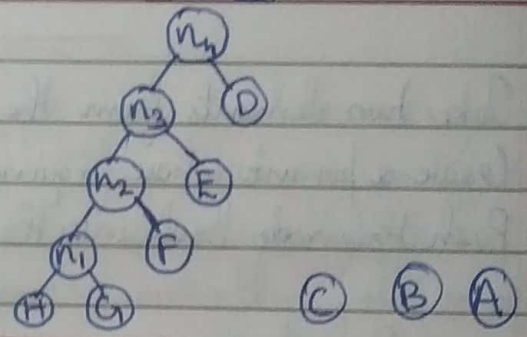


iv) Take two elements from the priority queue: $(n_3, f_{n_3}), (D, f_D)$
Create a parent whose frequency is the sum of those of its children: $(n_4, f = f_{n_3} + f_D = 1/8)$
Push this node back into the queue.

Priority queue

$(n_4, f_{n_4} = 1/8)$ n_4 is the parent of G & H
$(C, f_C = 1/8)$ n_2 is the parent of n_1 & F
$(B, f_B = 1/4)$ n_3 is the parent of n_2 & E
$(A, f_A = 1/2)$ n_4 is the parent of n_3 & D

Tree

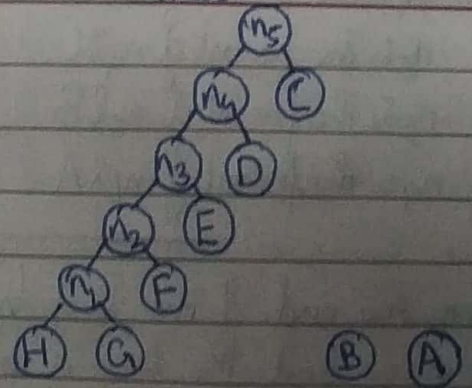


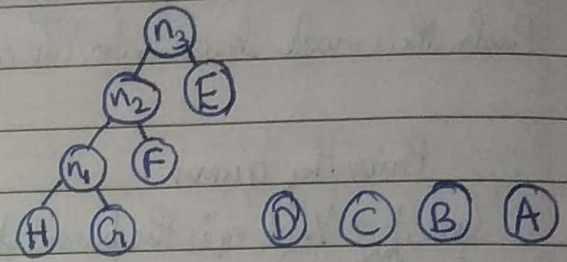
v) Take two elements from the priority queue: $(n_4, f_{n_4}), (C, f_C)$
Create a parent whose frequency is the sum of those of its children: $(n_5, f = f_{n_4} + f_C = 1/4)$
Push this node back into the queue.

Priority queue

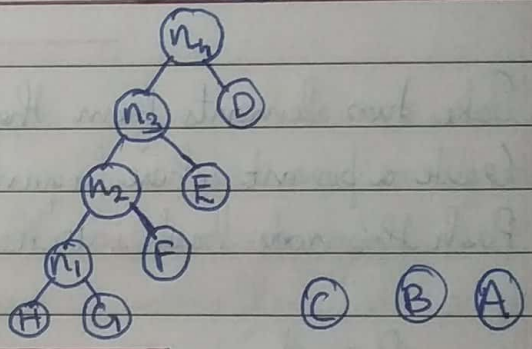
$(n_5, f_{n_5} = 1/4)$ n_4 is the parent of G & H
$(B, f_B = 1/4)$ n_2 is the parent of n_1 & F
$(A, f_A = 1/2)$ n_3 is the parent of n_2 & E
n_4 is the parent of n_3 & D
n_5 is the parent of n_4 & C

Tree.

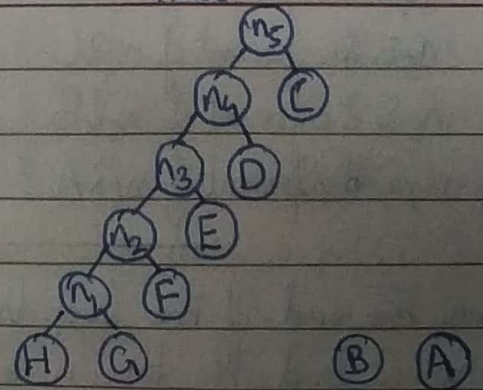


Priority queue	Tree
$(n_3, f_{n_3} = 1/16)$ n_4 is the parent of G & H	
$(D, f_D = 1/16)$ n_2 is the parent of n_1 & F	
$(C, f_C = 1/8)$ n_3 is the parent of n_2 & E	
$(B, f_B = 1/4)$	
$(A, f_A = 1/2)$	

iv) Take two elements from the priority queue: $(n_3, f_{n_3}), (D, f_D)$
Create a parent whose frequency is the sum of those of its children: $(n_4, f = f_{n_3} + f_D = 1/8)$
Push this node back into the queue.

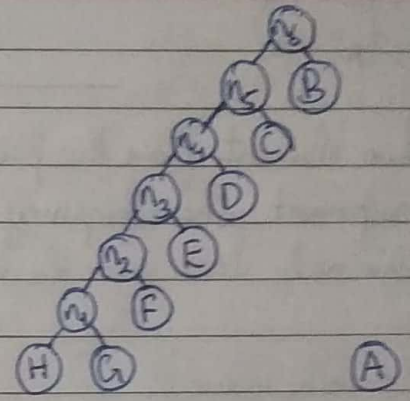
Priority queue	Tree
$(n_4, f_{n_4} = 1/8)$ n_4 is the parent of G & H	
$(C, f_C = 1/8)$ n_2 is the parent of n_1 & F	
$(B, f_B = 1/4)$ n_3 is the parent of n_2 & E	
$(A, f_A = 1/2)$ n_4 is the parent of n_3 & D	

v) Take two elements from the priority queue: $(n_4, f_{n_4}), (C, f_C)$
Create a parent whose frequency is the sum of those of its children: $(n_5, f = f_{n_5} + f_C = 1/4)$
Push this node back into the queue.

Priority queue	Tree.
$(n_5, f_{n_5} = 1/4)$ n_1 is the parent of G & H	
$(B, f_B = 1/4)$ n_2 is the parent of n_1 & F	
$(A, f_A = 1/2)$ n_3 is the parent of n_2 & E	
n_4 is the parent of n_3 & D	
n_5 is the parent of n_4 & C	

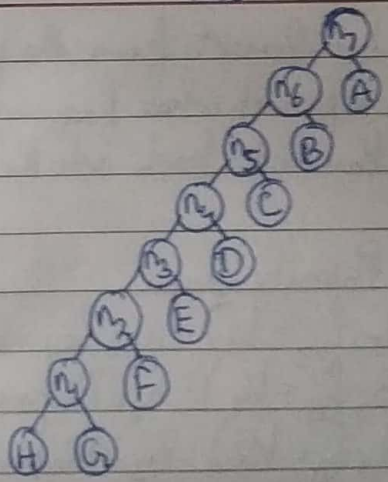
vi) Take two elements from the priority queue: (n_5, f_{n_5}), (B, f_B)
Create a parent whose frequency is the sum of those of its children: $(n_6, f = f_{n_5} + f_B = 1/2)$
Push this node back into the queue.

Priority queue	Tree
$(n_6, f_{n_6} = 1/2)$	n_6 is the parent of G & H
$(A, f_A = 1/2)$	n_5 is the parent of n_4 & F
	n_4 is the parent of n_3 & E
	n_3 is the parent of n_2 & D
	n_2 is the parent of n_1 & C
	n_1 is the parent of n_5 & B



vii) Take two elements from the priority queue: (n_6, f_{n_6}), (A, f_A)
Create a parent whose frequency is the sum of those of its children: $(n_7, f = f_{n_6} + f_A = 1)$
Push this node back into the queue.

Priority queue	Tree
$(n_7, f_{n_7} = 1)$	n_7 is the parent of G & H
	n_6 is the parent of n_5 & F
	n_5 is the parent of n_4 & E
	n_4 is the parent of n_3 & D
	n_3 is the parent of n_2 & C
	n_2 is the parent of n_1 & B
	n_1 is the parent of n_6 & A



Since the end of the loop has been reached (8 characters, 7 iterations), n_7 is returned and the tree is as above. The prefix code is as follows:

A = 1 B = 01 C = 001 D = 0001 E = 00001
F = 000001 G = 0000001 H = 0000000

Average Bit length of the Code = (Bitlength(A) + Bitlength(B) + Bitlength(C) + Bitlength(D) + Bitlength(E) + Bitlength(F) + Bitlength(G) + Bitlength(H))/8
= (1+2+3+4+5+6+7+7)/8 = 35/8 = 4.375 //

4. Describe an efficient algorithm that, given a set $\{x_1, x_2, \ldots, x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

quick select partition pick

1) sort(points)
2) S = ϕ
3) for i = 0 to length(points) - 1
   3.1) if (points[i] ∉ S.last_added()) OR empty(S).
      3.1.1) S.append([points[i], points[i]+1])
4) return S.

To show that our greedy choice has an optimal solution, consider any optimal solution, where $x_1 \in [p, p+1]$. Since there are no elements to the left of x_1, the interval can be shifted to begin at x_1, as $[x_1, x_1+1]$, and will still cover all the points it covered initially if not more. Thus, the greedy choice is justified.

For optimal substructure, remove all the values in $[x_1, x_1+1]$ from points and solve the remaining subproblem S'. An optimal solution to the problem, S will be $S' \cup [x_1, x_1+1]$. To show this, let the left most point in the subproblem be x_i. We know $x_i \notin [x_1, x_1+1]$, ∴ there is no solution to the subproblem with an interval that includes x_1 ($x_i - x_1 > 1$). Thus the solution to S must be such that $|S| \geq |S'| + 1$. The solution we have constructed has $|S| = |S'| + 1$, and is hence ~~the most~~ optimal. Thus the algorithm is correct.

5. There exists a $O(n)$-time deterministic algorithm (M) for finding median of n given numbers. Using this algorithm as a subroutine, design a $O(n)$-time deterministic algorithm for solving the fractional knapsack problem (items are $(I_i)_{i=1}^n$, weight of items are $(w_i)_{i=1}^n$, profit of items are $(p_i)_{i=1}^n$, and knapsack capacity is W), and also prove its time complexity.

The algorithm here uses a quicksort style partitioning technique. The recursion is simple, partition around a pivot and recursively search through the relevant partition. Both partitions needn't be searched through, let the first partition contain all those elements with a p/w higher than that of the pivot. Let the second partition contain all those elements lesser in value than the ~~median~~ pivot. If the total weight of the first partition is more than that of the knapsack, then search only this partition. If it is less, subtract this total weight from that of the knapsack (till the knapsack with all the elements in this partition) and then search the second partition for the difference.

This is a randomized algorithm, use the deterministic median as the pivot for a deterministic algorithm.

Fractional-Knapsack(a, W) $\longrightarrow T(n)$

1) med = Median(a, p/w) $\longrightarrow \Theta(n)$
2) $R_1, R_2, R_3 = \phi$, $w_1, w_2, w_3 = 0$ $\longrightarrow \Theta(1)$
3) n = length(a) $\rightarrow \Theta(n)$
4) for i = 0 to n-1 $\rightarrow n \times$
   4.1) if (a[i].p/a[i].w) <= med $\rightarrow \Theta(1)$
      4.1.1) R_1.append(a[i]) $\rightarrow \Theta(1)$
      4.1.2) $W_1 = W_1 + a[i].w$ $\rightarrow \Theta(1)$
   4.2) else if (a[i].p/a[i].w) == med $\rightarrow \Theta(1)$
      4.2.1) R_2.append(a[i]) $\rightarrow \Theta(1)$
      4.2.2) $W_2 = W_2 + a[i].w$ $\rightarrow \Theta(1)$
   4.3) else
      4.3.1) R_3.append(a[i]) $\longrightarrow \Theta(1)$
      4.3.2) $W_3 = W_3 + a[i].w$ $\longrightarrow \Theta(1)$
5) if $W \leq W_1$ $\longrightarrow \Theta(1)$
   5.1) return Fractional-Knapsack(R_1, W) $\rightarrow T(n/2)$

$+T(n/2)$
$\Theta(n)$.

6) else if $W_1 < W \leq W_1 + W_2$ ⟶ $\Theta(1)$
6.1) ~~Solution = Fractional-Knapsack(~~ ⟶ $\Theta(n)$
6.1) $W = W - W_1$, $i = 0$, $n = \text{length}(R_2)$
6.2) ~~n = length~~ while $(W > 0)$ AND $(i < n)$ ⟶ $n \times \Theta(1) \times$
~~6.2.1) R_1.append(a[~~
6.2.1) if $W < R_2[i].w$ ⟶ $\Theta(1)$
6.2.1.1) $R_2[i].f = R_2[i].w / W$ ⟶ $\Theta(1)$
6.2.1.2) R_1.append$(R_2[i])$ ⟶ $\Theta(1)$
6.2.1.3) $W = W - W$ ⟶ $\Theta(1)$
6.2.2) else ⟶ $\Theta(1)$
6.2.2.1) R_1.append$(R_2[i])$ ⟶ $\Theta(1)$
6.2.2.2) $W = W - R_2[i].w$ ⟶ $\Theta(1)$
6.2.3) $i = i + 1$. ⟶ $\Theta(1)$
6.3) return R_1 ⟶ $\Theta(1)$
7) else if $W_1 + W_2 < W \leq W_1 + W_2 + W_3$ ⟶ $\Theta(1)$
7.1) return $R_1 \cup R_2 \cup$ Fractional-Knapsack$(R_3, W - W_1 - W_2)$ ⟶ $T(n/2)$
8) else
8.1) return "Knapsack is too big" ⟶ $\Theta(1)$.

Complexity of steps 1 → 4 = $\Theta(n) + \Theta(1) + \Theta(n) + n \times (\Theta(1) + \Theta(1) + \Theta(1) + \Theta(1))$ ⟶ Worst case
$= \Theta(n) + n\Theta(1) = \Theta(n)$

If step 8 is entered, final complexity = $\Theta(n) + \Theta(1) + \Theta(1) + \Theta(1) + \Theta(1)$
$= \Theta(n)$

If step 6 is entered, final complexity = $\Theta(n) + \Theta(1) + \Theta(1) + \Theta(n) + n \times \Theta(1) \times (\Theta(1) + \Theta(1) + \Theta(1) + \Theta(1)) + \Theta(1)$ ⟶ Worst case
$= \Theta(n) + n \times \Theta(1) \times \Theta(1) = \Theta(n)$

~~this step needs to be executed n times (worst case) but not $\Theta(n)$~~ ⟶ Since $\Theta(n) \Rightarrow O(n)$, change all Θ to O

If step 7 is entered, final complexity = $\Theta(n) + \Theta(1) + \Theta(1) + \Theta(1) + T(n/2)$
$T(n) = T(n/2) + \Theta(n) = \Theta(n) + \Theta(n/2) + T(n/4) = \Theta(n) + \Theta(n/2) + \Theta(n/4) + T(n/8)$
$= \Theta(n) + \Theta(n/2) + \Theta(n/4) + \Theta(n/8) \ldots\ldots = \Theta(n(1 + 1/2 + 1/4 + \ldots)) = \Theta(2n) = \Theta(n)$

Similarly for step 1.

∴ Worst case complexity for this algorithm is $O(n)$

6. Show that no compression scheme can expect to compress a file of randomly chosen 8-bit characters by even a single bit.

?? Verify.

Let the the compression scheme be described by a pair of algorithms (C, D). Let us describe these algorithms as functions. Let U be the set of all uncompressed files with n randomly chosen 8 bit characters. The total number of bits in such a file is $n \times 8$, and the number of total such files $|U| = 2^{n \times 8} = 256^n$. If there is a compression scheme that compresses any arbitrary file by k bits, any file in the set S of compressed files has $n \times 8 - k$ bits, and thus the number of such files, $|S| = 2^{8n-k} = 2^{8n}/2^k$.

$\therefore |U| = 2^{8n} = 256^n$, $|S| = 2^{8n}/2^k = 256^n/2^k$.

Thus, C needs to take a file from U and return a file from S.

$C : U \to S$.

If C is ~~the number~~ expressed as a set of ordered pairs, the number of elements in C, $|C| = 256^n$ (one ~~encryption~~ compression for each random file in U)

Since D is supposed to decompress a file in S to ~~return~~ a file in U,

$D : S \to U \Rightarrow |S| \geq |U|$ (There should be a compression for atleast every ~~$|U| \geq 256^n/2^k$ (one decompression for every compressed file in S).~~ decompressed file).

~~But C and D are supposed to be inverses of each~~

$\therefore 256^n \leq \frac{256^n}{2^k} \Rightarrow 2^k \leq 1 \Rightarrow k \leq \log_2 1 \Rightarrow k \leq 0$ (positive value)

∴ Thus such random files cannot be compressed.

The argument can be trivially expanded to files of length almost n & ~~encryptions~~ compressions almost $8n - k$ bits.

7. Suppose we have an optimal prefix code on a set $C = \{0, 1, \ldots, n-1\}$ of characters and we wish to transmit this code using as few bits as possible. Show how to represent any optimal prefix code on C using only $2n - 1 + n\lceil \log n \rceil$ bits. Make a binary code for the optimal prefix code of problem 2(a).

Lemma 1:
The binary prefix tree with the maximum length character codes for a given set of characters is the one in which there is exactly one leaf and one internal node in the leaf each level, except for the first and the last. The first level has only the root, and the last has only two leaves.

Have this in the exam.

Proof: Consider any tree T that is not of this form. In such a tree, there must atleast one level with exactly 2 internal nodes and no leaves. Let this level be i, and let the two internal nodes be 'a' and 'b'. Let 'b' be the node whose subtree has the deepest element in the tree, and let this element be x.

Consider the subtree rooted at 'a'. Consider the sum of depths of all leaf nodes of this subtree wrt to 'a'. Let this be D_a. Since 'a' is internal in an optimal binary tree, $D_a \geq 2$. If there are n_a leaf nodes in this subtree, and n_x elements in the other subtree, swap x with the subtree rooted at 'a'. Call the new tree T'. Then

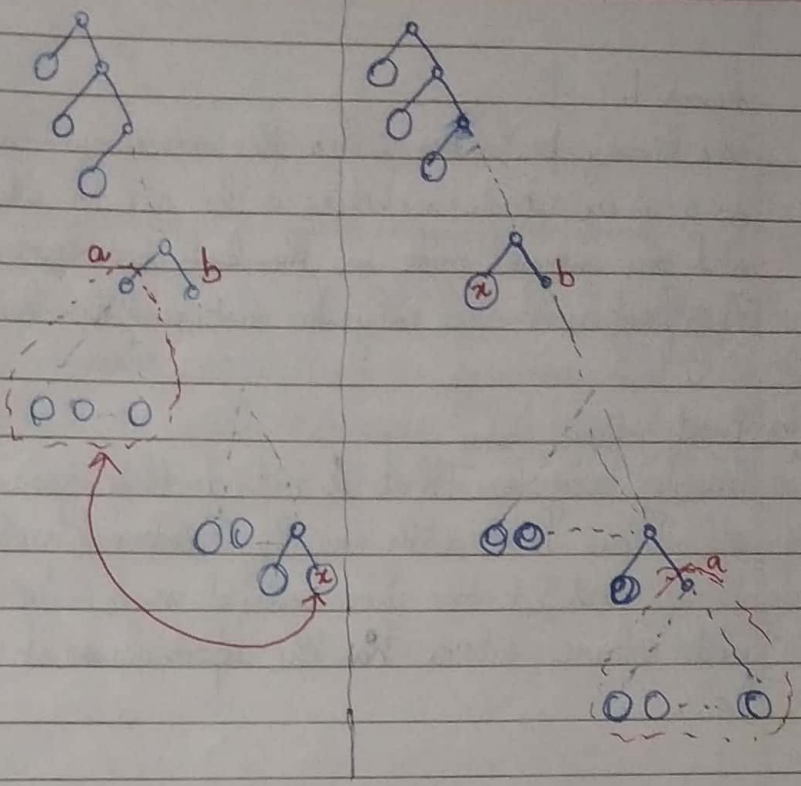
$$\begin{aligned} D_{T'} &= D_T - n_a d_T(a) - D_a - d_T(x) + n_a d_{T'}(a) + D_a + d_{T'}(x) \\ &= D_T - n_a d_T(a) - D_a - d_T(x) + n_a d_T(x) + D_a + d_T(a) \\ &= D_T + (n_a - 1)(d_T(x) - d_T(a)) \end{aligned}$$

$n_a > 2$ ('a' is internal in an optimal tree).

x is the deepest leaf and a is an internal node in T $\Rightarrow d_T(x) > d_T(a)$.

$\therefore D_{T'} > D_T$

Therefore, $D_{T'}$ is maximum only according to Lemma 1.



Therefore, for n characters, the total no of bits required for all characters
$= 1 + 2 + 3 + \dots + n-1 + n-1 = \frac{(n-1)n}{2} + (n-1) = \frac{(n-1)(n+2)}{2}$

prove this { Consider any full binary tree. It can be encoded in n bits, where n is the number of nodes. This is very simple, ~~###~~ Just mark all internal nodes as 0, all leaf nodes as 1 and perform a pre order traversal.

For any optimum code, there are n leaves & n-1 internal nodes. Thus, the structure of the tree is represented by 2n-1 bits. Now, to associate every leaf with a character, each of n characters can be represented in $\lceil \log n \rceil$ bits, any permutation of these can be represented in $n\lceil \log n \rceil$ bits. The permutations of the characters that appear in the leaf nodes as the leaves are ordered in the preordered traversal thus can be specified in $n\lceil \log_2 n \rceil$ bits. Thus, the complete tree needs $2n-1+n\lceil \log_2 n \rceil$ bits to be specified.

8. Suppose that a data file contains a sequence of 8-bit characters such that all 256 characters are about equally common: the maximum character frequency is less than twice the minimum character frequency. Prove that Huffman coding in this case is no more efficient than using an ordinary 8-bit fixed-length code.

1) The binary tree for a fixed 8-bit code with 256 characters is the complete binary tree with 256 leaves (since all the leaves are at the same depth, 8).

2) If the priority queue in Huffman's algorithm functions as a regular (FIFO queue) queue, then with 2^n characters, the tree produced is the complete binary tree with 2^n leaves.

Proof:

After 2^{n-1} times, any two leaves are taken and a common parent is created, which is pushed to the end. Since the queue is a regular queue, all leaf nodes are paired before any parent nodes are paired.

prove by induction

For 2^{n-2} times, all subtrees of depth 1 are paired.

⋮

For 2^{n-k} times, all subtrees of depth $k-1$ are paired.

⋮

In the last iteration, the 2^{n-n} th iteration, all subtrees of depth $n-1$ are paired and only one subtree of depth n remains.

3) Since no element in the priority queue occurs more than twice as much as any other, the root of any subtree of depth k has a higher frequency than any subtree of depth $k-1$. Thus, and any subtree of depth k added after any other such subtree of same depth has a higher frequency than the earlier tree. Thus, the priority queue acts just like a FIFO queue.

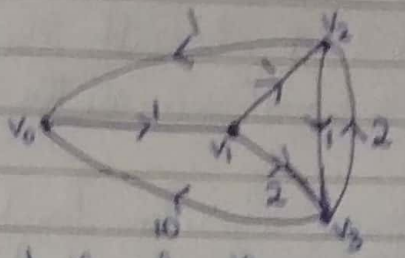
Prove by induction

prove this

Hence the Huffman code is a regular fixed length code.

9. Let G be a directed graph with n vertices. Let $l(u, v)$ be the (non-negative) length of the directed edge $u \to v$. A path starting at a given vertex v_0, going through every other vertex exactly once, and finally returning to v_0 is called a *tour*. The length of a tour is the sum of the lengths of the edges on the path defining the tour. We are concerned with finding a tour of minimum length. A greedy way to construct such a tour is: let (P, v) represent the path so far constructed; it starts at v_0 and ends at v. Initially P is empty and $v = v_0$, if all vertices in G are on P, then include the edge $v \to v_0$ and stop; otherwise include an edge $v \to w$ of minimum length among all edges from v to a vertex w not on P. Show that this greedy method does not necessarily generate a minimum-length tour.

Consider the graph:



$G = (\{v_0, v_1, v_2, v_3\}, \{(v_0 \to v_1, 1), (v_0 \to v_2, 1), (v_1 \to v_3, 2), (v_0 \to v_3, 1), (v_3 \to v_2, 2), (v_2 \to v_0, 1), (v_3 \to v_0, 10)\})$

V E

According to the algorithm,

1) $(P = \phi, v = v_0)$
2) $(P = \{(v_0 \to v_1, 1)\}, v = v_1)$
3) $(P = \{(v_0 \to v_1, 1), (v_1 \to v_2, 1)\}, v = v_2)$
4) $(P = \{(v_0 \to v_1, 1), (v_1 \to v_2, 1), (v_2 \to v_3, 1)\}, v = v_3\}$
5) $(P = \{(v_0 \to v_1, 1), (v_1 \to v_2, 1), (v_2 \to v_3, 1), (v_3 \to v_0, 10)\}, v = v_0\}$

Algorithm stops.

The length of this path $P = 1+1+1+10 = 13$.

However, the path $P' = \{(v_0 \to v_1, 1), (v_1 \to v_3, 2), (v_3 \to v_2, 2), (v_2 \to v_0, 1)\}$ has length $1+2+2+1 = 6$ //

Thus, the algorithm is incorrect.

10. Consider the following scheduling problem: n jobs are given as input. Job j ($1 \le j \le n$) has a processing time p_j ($p_j > 0$) and a non-negative weight w_j ($w_j \ge 0$). We must construct a schedule for these jobs on a single machine such that at most one job is processed at each point in time, and each job must be processed nonpreemptively; that is, once a job begins to be processed, it must be processed completely before any other job begins its processing. The objective is to find a schedule that minimizes the weighted sum of completion times: $\sum_{j=1}^{n} w_j C_j$. Suppose that jobs are indexed such that $\frac{w_1}{p_1} \ge \frac{w_2}{p_2} \ge \ldots \ge \frac{w_j}{p_j} \ge \ldots \ge \frac{w_n}{p_n}$. Then prove that it is optimal to schedule the jobs in the order $(1, 2, \ldots, j, \ldots, n)$ (job 1 first, job 2 second, and so on).

Firstly, C_j is the sum of all p_i, $i \le j$ including j that have finished before it.

To justify the greedy choice, consider any optimal solution, as any permutation of $(1, 2, \ldots, n)$.

~~Consider any two jobs i and j such that $w_i/p_i \geq w_j/p_j$ and $C_i \geq C_j$~~
~~Since~~

Consider the job n, with minimum w_n/p_n, which in the optimal solution has index i. Consider also in the n^{th} position some job j with $w_j/p_j \geq w_n/p_n$.

We have $C_j > C_n$. Let this schedule be called S, and let S' be the schedule where j and n are interchanged. Let W_S be the weighted sum of completion times in schedule S. Let B be the set of all jobs finished before n, and A be the set of all jobs completed after n, ~~not~~ including it, in schedule S. or not including j

$$W_S = \sum_{i=1}^{n} w_i C_i = \sum_{i \in B} w_i C_i + \sum_{i \in A} w_i C_i + w_j C_j + w_n C_n$$

$$W_{S'} = \sum_{i=1}^{n} w_i C_i' = \sum_{i \in B} w_i C_i' + \sum_{i \in A} w_i C_i' + w_j C_j' + w_n C_n'$$

In S', all jobs in B have the same completion time as in S, and for all jobs i in A, $C_i' = C_i - p_n + p_j$. Also, $C_n' = C_j$, $C_j' = C_j - p_j + p_n$

$$\therefore W_{S'} = \sum_{i \in B} w_i C_i + \sum_{i \in A} w_i (C_i - p_n + p_j) + w_j (C_j - p_j + p_n) + w_n C_j$$

~~$\therefore W_{S'} - W_S = \sum_{i \in A} (w_i C_i - w_i p_n + w_i p_j - w_i C_i) = \sum_{i \in A} w_i p_j - w_i p_n$~~

$$W_S - W_{S'} = \sum_{i \in A} w_i (C_i' - C_i) + w_j (C_j - C_n + p_j - p_n) + w_n (C_n - C_j)$$

$$= \sum_{i \in A} w_i (p_j - p_n) + (w_j - w_n)(C_j - C_n) + w_j (p_j - p_n).$$

$$= \sum_{i \in A \cup \{j\}} w_i (p_j - p_n) + (w_j - w_n)(C_j - C_n)$$

But $C_j - C_n = \sum_{i \in A \cup \{j\}} p_i$

$$\therefore W_S - W_{S'} = \sum_{i \in A \cup \{j\}} (w_i (p_j - p_n) + (w_j - w_n) p_i)$$