Question One

c) $T(n) = 3T(n/3) + 1$, as function decomposes $T(n)$ into three subproblems of size $n/3$ and recursively calls itself to solve those sub-problems, as seen in the assignment of k on line 5, followed by 3 recursive calls from lines 6-8.

Question Two

a) i)
```
FIND_SMALLEST_SUM(A): // one-based indexing
    A = sort(A)                        | O(n lg n)
    return array <A[1], A[2]>          | 1
```
This algorithm uses Mergesort to sort A as it has the best worst-case time complexity of any sorting algo and since the last line is constant time, the function has a run-time of $O(n \lg n)$

ii)
```
FIND_SMALLEST_SUM(A): // one-based indexing
    smallest, smallestIdx = ∞, -1                          | 1
    secSmallest = ∞                                        | 1
    for i in range(len(A)):                                | n
        if A[i] < smallest:                                | 1
            smallest = A[i]                                | 1
            smallestIdx = i                                | 1
    for i in range(len(A)):                                | n
        if A[i] < secSmallest and i != secSmallest         | 1
            secSmallest = A[i]                             | 1
    return array <smallest, secSmallest>                   | 1
```
$\therefore O(1 + 1 + n + 1 + 1 + 1 + n + 1 + 1 + 1)$

$= O(n)$

b) i)

```
FIND_MATCHES(X, Y):  // 1-based indexing
    X = sort(X)                                   | nlgn
    Y = sort(Y)                                   | nlgn
    xIdx = 1                                      | 1
    yIdx, numMatches = 1, 0                       | 1
    WHILE xIdx <= len(X) and yIdx <= len(Y)       | n
        if X[xIdx] == Y[yIdx]:                    | 1
            numMatches++                          | 1
            xIdx++                                | 1
            yIdx++                                | 1
        elif X[xIdx] < Y[yIdx]:                   | 1
            xIdx++                                | 1
        elif Y[yIdx] < X[xIdx]                    | 1
            yIdx++                                | 1
    return numMatches                             | 1
```

* NOTE: this algorithm uses Mergesort as it has the best worst-case time complexity of any comparison-based algorithm ($O(n\lg n)$), ∴

$O(n\lg n + n\lg n + n + 11)$

$= O(n\lg n)$

ii) No, as substrings are matched based on not only their content, but their ordering. and since we seek to find matches regardless of ordering, LCS will not work. e.g.

$k = 3$

$X = \langle 3, 1, 2, 3, 1, 2 \rangle$

$Y = \langle 1, 1, 2, 3, 3, 2 \rangle$

Applying LCS would yield a result of "1, 2, 3, 2", therefore 4 matches. However this is incorrect as there are 6 matches

iii) No, because we need to iterate over both sets (at minimum) in order to find matches, an order of $O(n)$ operation.

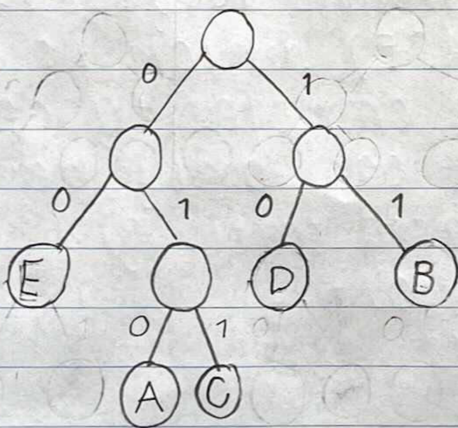
QUESTION THREE

a) i)

char	freq
A	5
B	15
C	6
D	14
E	10

$Q = \langle A, C, E, D, B \rangle$

$\therefore$



A = 010
B = 11
C = 011
D = 10
E = 00

QUESTION ONE

$\cdot\ T(n) = 5T(n/6) + n$

$- a = 5,\ b = 6,\ f(n) = n$

$1:\ n^{\log_6 5} = n^{0.8982}$

$f(n) > n^{0.8982}$

$n > n^{0.8982}$ $\therefore$ Case 3

$2:\ f(n)/n^{\log_b a} = n/n^{0.8982} \geq n^{\varepsilon},\ \varepsilon \approx 0.1$

$3:\ 5(n/6)$

$\leq (5/6)n$

$\leq cf(n),\ c \geq 5/6$

$\therefore T(n) = \Theta(n \lg n)$

$\therefore$ Yes, as the time complexity is $\Theta(n \lg n)$, the best possible for comparison-based sorting

Depends, I would need some knowledge about it space complexity before making that decision.

- $T(n) = 4T(n/3) + n$
  - $a = 4$, $b = 3$, $f(n) = n$

  1: $n^{\log_3 4} = n^{1.2619}$

  $f(n) < n^{1.2619}$

  $n < n^{1.2619}$ $\therefore$ Case 1

  2: $f(n) = O(n^{\log_3 4 - 1})$

  $= O(n^{\log_3 3})$

  $= O(n)$

  $\therefore T(n) = O(n)$

  $\therefore$ No, as the runtime is given as $\Theta(n)$ ~~(it's WORST CASE)~~ which is impossible as the time complexity of a sorting algo at best can only be $\Omega(n \lg n)$ (comparison-based sorting)

b) $\cdot T(n) = T(2n/10) + T(8n/10) + n$

- $T(n) \leq cn\lg n$

  $T(n) = T$

A B C D M P S

Question Three

b) i) Yes, as it will always execute because since the for loop condition is $s < n-m$, s will always be less than $n-m$ (i.e. $s < n-m$)

c) i) · Finding MCSTs
∴ I will use the generic algorithm from the Lecture 6 slides

ii) 1: Choose A, V = <B, C, D, M, P, S>
2: Candidate edges = (A,B), (A,C), (A,D), (A,M), (A,P), (A,S)
* 3: Choose (A,M), V = <B, C, D, P, S>
4: Candidate edges = (A,B), (A,C), (A,D), (A,P), (A,S), (M,S)
* 5: Choose edge (M,S), V = <B, C, D, P>
6: Candidate edges = (A,B), (A,C), (A,D), (A,P), (A,S), (S,B)
* 7: Choose edge (S,B), V = <C, D, P>
8: Candidate edges = (A,B), (A,C), (A,D), (A,P), (A,S), (B,C)
* 9: Choose edge (B,C), V = <D, P>
10: Candidate edges = (A,B), (A,C), (A,D), (A,P), (A,S), (C,D)
* 11: Choose edge (C,D), V = <P>
12: Candidate edges = (A,B), (A,C), (A,P), (A,S), (D,P)
* 13: Choose edge (D,P), V = <>

∴ Dist = 9 + 7 + 8 + 17 + 20 + 32
= 93

∴ cost = 0.50 × 93
= \$46.50

Question Four

a) i) • KNAPSACK(w, p, c): // One-based indexing

```
    maxProfit = 0                                    | 1
    i = 1                                            | 1
    remaining_capacity = c                           | 1
    WHILE w[i] ≤ remaining_capacity                  | n
    and i <= len(w):
        maxProfit += p[i]                            | 1
        remaining_capacity -= w[i]                   | 1
        i++                                          | 1
    return maxProfit                                 | 1
```

• $\therefore O(n + 7)$
$= O(n)$

• W = [1 2 4 5 7]
p = [8 6 5 3 1]
1: maxProfit = 8, rem_cap = 7
2: maxProfit = 14, rem_cap = 5
3: maxProfit = 19, rem_cap = 1
4: return maxProfit

c) i) Parallel_search(x, A):

```
for all Pi do in parallel: // 1 ≤ i ≤ n/2
    if A[i] = x then
        index ← i
    elif A[i + n/2] = x then
        index ← i + n/2
```

ii) $C(n) = P(n) \times T(1)$
$= O(n)$

$T^*(n) = n$

$\therefore$ since $C(n) = T^*(n)$, algo is cost optimal

b) i) $A = 5 \times 3$, $B = 3 \times 1$, $C = 1 \times 4$, $D = 4 \times 6$

a) iii) ...