Design and Analysis of Algorithms

Assignment #3

Sri Devi Mallipudi (16251191)
Pragathi Thammaneni (16230695

1) Describe an O(n) algorithm that given a set of n distinct numbers and a positive integer $k <= n$, determines the k numbers is S that are closest to the median of S.

Sol:

```
select (A, p, r, i)
    if p == r
        return A[p]
  x = median(A, p, r)
  q = partition (A, p, r, x)
  k = q - p + 1
  if   i == k
      return A[q]
  else if  i < k
      return select (A, p, q-1, i)
  else
      return select (A, q+1, r, i-k)

median (A, p, r)
    if r - p < 5
        return partition (A, p, r)
    for i ← p to r
        SR = i+4
        if SR > r
            SR = r
```

```
med5 = partitions (A, i, SR)
swap A[med5] ↔ A[p + [(i-p)/5]]
return select (A, p, p + [(r-p)/5] - 1, p + (r-p)/10)

partition (A, p, r, x)
    pv = A[x]
    swap A[x] ↔ A[r]
    SI = p
    for i ← p to r-1
        if A[i] < pv
            swap A[SI] ↔ A[i]
            SI++
    swap A[r] ↔ A[SI]
    return SI
```

Complexsity : $O(n)$

2) Find an optimal parenthesization of a matrix chain multiplication for the following matrices:

A	B	C	D	E
7×10	10×9	9×5	5×12	12×6

sol:-

	A	B	C	D	E
A	A 0 7×10	AB 630 7×9	ABC 800 7×5	ABCD 1220 7×12	ABCDE 1370 7×6
B	–	B 0 10×9	BC 450 10×5	BCD 1050 10×12	BCDE 1110 10×6
C	–	–	C 0 9×5	CD 540 9×12	CDE 630 9×6
D	–	–	–	D 0 5×12	DE 360 5×6
E	–	–	–	–	E 0 12×6

m[i, j]

k- table

–	1	1	3	3
–	–	2	3	3
–	–	–	3	3
–	–	–	–	4
–	–	–	–	–

S[i, j]

→ To obtain the matrix A from A, we do not need to multiply anything.

→ Hence, the value would be zero.

→ Similar to the case with B, C, D and E

→	A	B	C	D	E
	7×10	10×9	9×5	5×12	12×6

No. of multiplications for:

$AB = 7\times10\times9 = 630$ $(k=1)$

$BC = 10\times9\times5 = 450$ $(k=2)$

$CD = 9\times5\times12 = 540$ $(k=3)$

$DE = 5\times12\times6 = 360$ $(k=4)$

→

$$\begin{tikzcd} & ABC \arrow[dl, no head] \arrow[dr, no head] & \\ A(BC) & & (AB)C \\ (k=1) & & (k=2) \end{tikzcd}$$

$A(BC) = 0+450+(7\times10\times5) = 800$

$(AB)C = 630+0+(7\times9\times5) = 945$

→ Since 800 is the minimum value among 800 and 945, we consider $A(BC) \Rightarrow k=1$

→

$$\begin{tikzcd} & BCD \arrow[dl, no head] \arrow[dr, no head] & \\ B(CD) & & (BC)D \\ (k=2) & & (k=3) \end{tikzcd}$$

$B(CD) = 0+540+(10\times9\times12) = 1620$

$(BC)D = 450+0+(10\times5\times12) = 1050$

→ Since 1050 is the minimum value among 1620 and 1050, we consider (BC)D ⇒ k=3

→ CDE

C(DE) (k=3) (CD)E (k=4)

C(DE) = 0 + 360 + (9×5×6) = 630

(CD)E = 540 + 0 + (9×12×6) = 1188

→ Since, 630 is the minimum value among 630 and 1188, we consider C(DE) ⇒ k=3

→ ABCD

A(BCD) (k=1) (AB)(CD) (k=2) (ABC)D (k=3)

A(BCD) = 0 + 1050 + (7×10×12) = 1890

(AB)(CD) = 630 + 540 + (7×9×12) = 1926

(ABC)D = 800 + 0 + (7×5×12) = 1220

→ Since 1220 is the minimum value among the three, we consider (ABC)D ⇒ k=3.

→ BCDE

B(CDE) (k=2) (BC)(DE) (k=3) (BCD)E (k=4)

B(CDE) = 0+ 630 + (10×9×6) = 1170

(BC)(DE) = 450+ 360 + (10×5×6) = 1110

(BCD)E = 1050 + 0 + (10×12×6) = 1770

→ Since 1110 is the minimum value among the three we consider (BC)(DE) ⟹ k=3

→

$$\begin{tikzcd}[column sep=small]
 & & ABCDE \arrow[dll, dash] \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
A(BCDE) \\ (k=1) & (AB)(CDE) \\ (k=2) & (ABC)(DE) \\ (k=3) & (ABCD)E \\ (k=4)
\end{tikzcd}$$

A(BCDE) = 0 + 1110+ (7×10×6) = 1530

(AB)(CDE) = 630+ 630+ (7×9×6) = 1638

(ABC)(DE) = 800 + 360+ (7×5×6) = 1370

(ABCD)E = 1220+ 0+ (7×12×6) = 1724

→ Since 1370 is the minimum value among all the 4, we consider (ABC)(DE) ⟹ k=3

→ Hence the paranthesization occurs as follows considering the k-table

(A)(BC) (DE)

⟹ (A (BC)) (DE)

3) Design an O(n²) dynamic programming algorithm to find a set of compatible activities such that the total amount of time the resource is used by these compatible activities is maximized.

i	1	2	3	4	5	6	7	8	9	10	11
S(i)	2	3	5	6	7	9	10	12	13	14	16
F(i)	6	5	7	10	8	13	16	14	14	18	20
L(i)	1	1	2	2	3	4	4	4	5	6	6
P(i)	∅	∅	2	1	3	5	5	5	6	9	9

Sol:

```
Compatibility (A[n] = [], i, S(i), F(i), L(i), P(i))
    if i == 1
        L = 1,
        P = ∅
        return A[i]
for i ← 2 to n
    for j ← i-1 to 1
        if S[i] < F[j]
            L[i] = L[i]
            if L == 1
                P = ∅
            else
                P(i) = P(i-1)
        return L(i), P(i)
    else
        $L(i) = L(i)+1$
        $P(i) = j$
        return $L(i), P(i)$
    end
end
$i == n$
while ($i != \phi$)
    add $i$ to $A[n]$
        $i = P(i)$
    return A;
```

Initial condition and sub-problems

$i = 1$

$\Rightarrow L = 1, \quad P = \phi$

$i = 2, \quad j = 1$

$\Rightarrow S(2) < F(1)$ True

$L = 1$

$P = \phi$

$i = 3, \quad j = 2$

$\Rightarrow S(3) < F(2)$ False

$L = 2$

$P = 2$

Complexsity

→ Since there are 2 nested 'for' loops, the time complexsity coould be $\underline{O(n^2)}$.