11

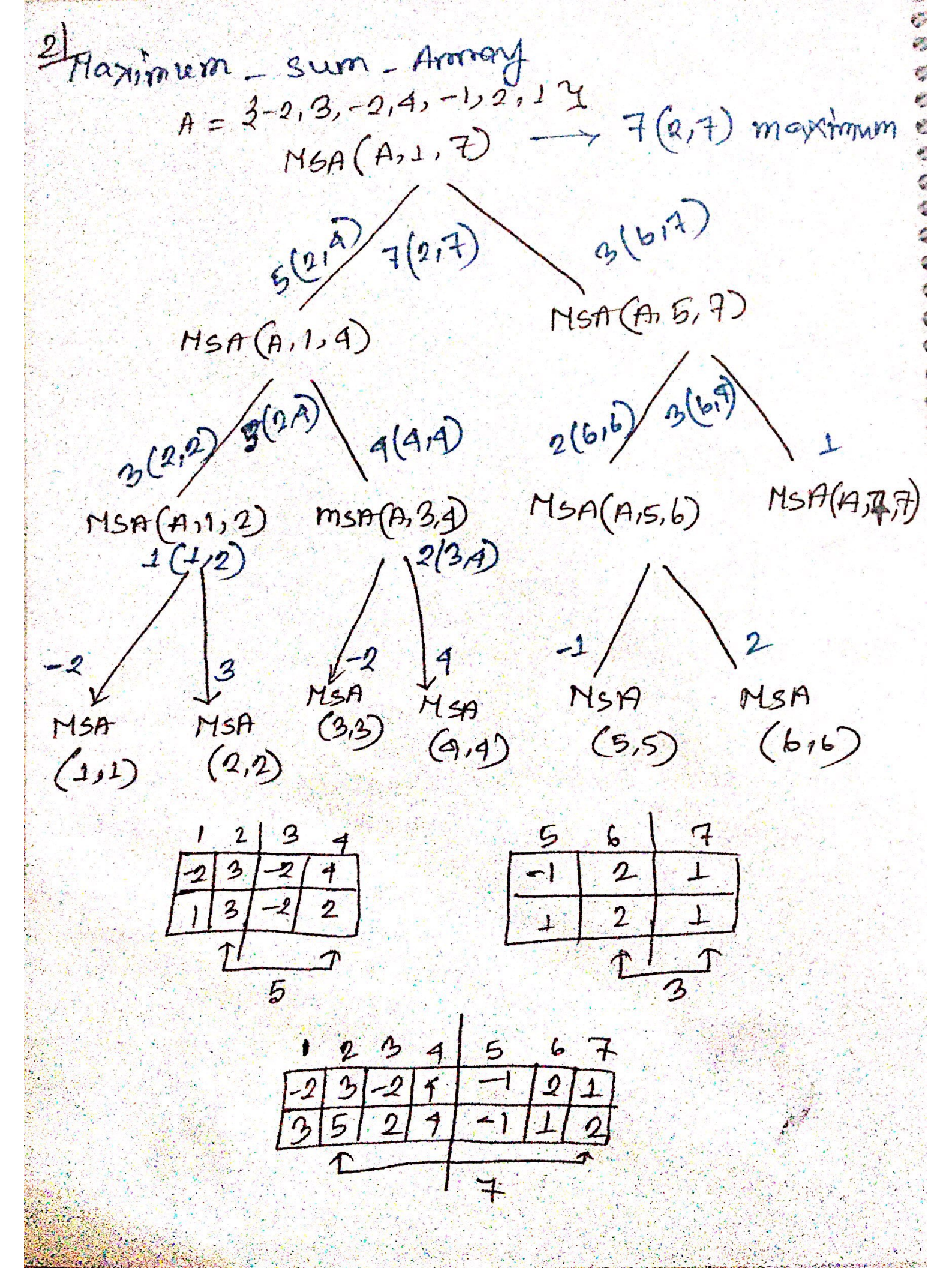
The time complexity of the Brute force method for solving the maximum-sum subarray problem is $O(n^3)$, where n is the size.

Algorithms like kadane's algorithm use in the divide-conquer approach, reduce the time complexity to $O(n \log n)$ by efficiently computing the maximum subarray sum without considering all possible subarrays separately. Resulting in a time complexity of $O(n \log n)$ significantly more efficient then the Brute force methods cubic complexity.

2| Maximum_sum_Array

A = {-2, 3, -2, 4, -1, 2, 1}

MSA(A, 1, 7) → 7(2,7) maximum



1	2	3	4
-2	3	-2	4
1	3	-2	2

5

5	6	7
-1	2	1
1	2	1

3

1	2	3	4	5	6	7
-2	3	-2	4	-1	2	1
3	5	2	4	-1	1	2

7

3]

L: 1 | 5 | 8 | 9 | 10 | 15 (i = 1)

R: 4 | 6 | 7 | 11 | 13 | 14 (j)

M:

step 1:

L: 1 | 5 | 8 | 9 | 10 | 15 (i = 1)

R: 4 | 6 | 7 | 11 | 13 | 14 (j)

M: 1

step: 2

1 | 5 | 8 | 9 | 10 | 15 (i)

4 | 6 | 7 | 11 | 13 | 14 (j)

1 | 4

step: 3

1 | 5 | 8 | 9 | 10 | 15 (i)

4 | 6 | 7 | 11 | 13 | 14 (j)

1 | 4 | 5

step: 4

1 | 5 | 8 | 9 | 10 | 15 (i)

4 | 6 | 7 | 11 | 13 | 14 (j)

1 | 4 | 5 | 6

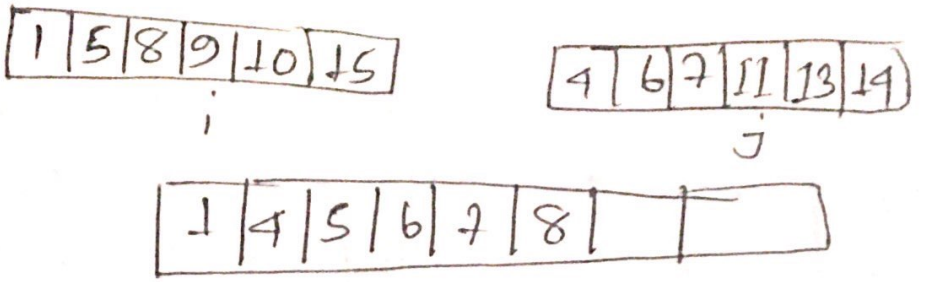
step: 5

1 | 5 | 8 | 9 | 10 | 15 (i)

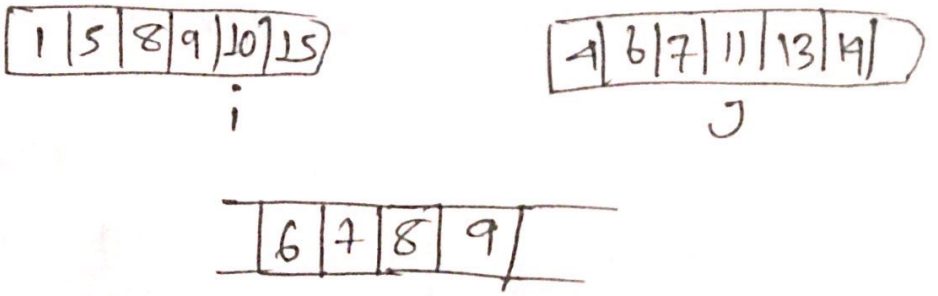
4 | 6 | 7 | 11 | 13 | 14 (j)

1 | 4 | 5 | 6 | 7

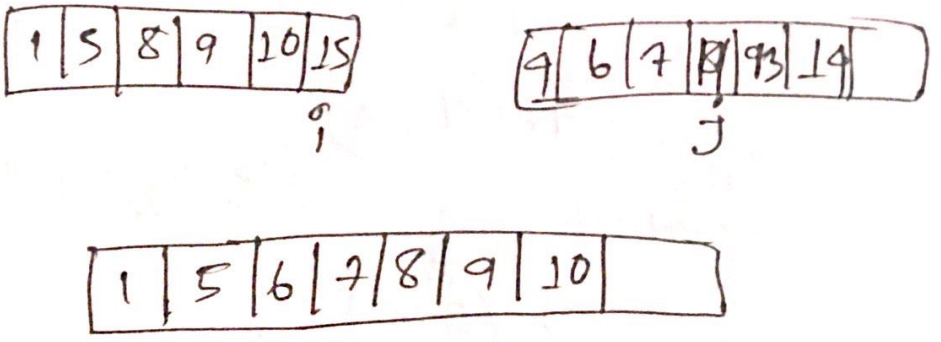
Step: 6



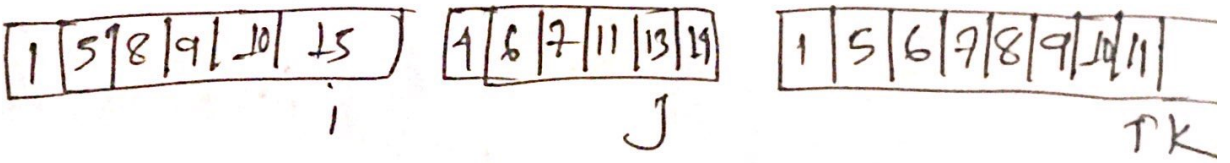
Step: 7



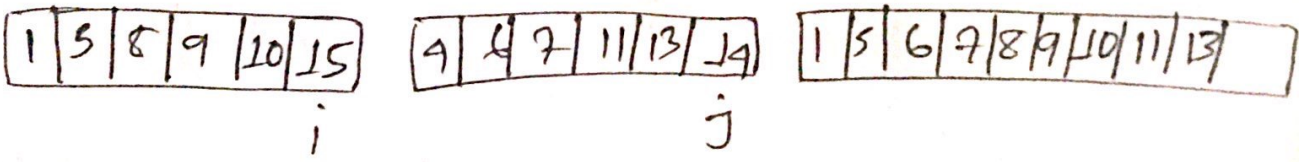
Step: 8:



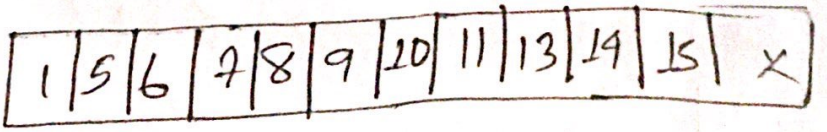
Step: 9



Step: 10:



Step: 11



4)

	T1	T2	T3	T4	T5	T6	T7	T8
st:	1000	840	850	1700	800	1300	1500	1200
end:	1030	1030	1040	2000	835	1800	1650	1380

→ sort in assending order of the ending times.

	T5	T1	T2	T3	T8	T7	T6	T4
st:	800	840	1000	850	1200	1500	1300	1700
end:	835	1030	1030	1040	1380	1650	1800	2000
	1	2	3	4	5	6	7	8

↑ [not fulfill the condition] ↑

N(1,8) {T5, T1, T8, T6, T4}

↓

N(2,8) {T1, T8, T6, T4}

↓

N(5,8) {T8, T6, T4}

↓

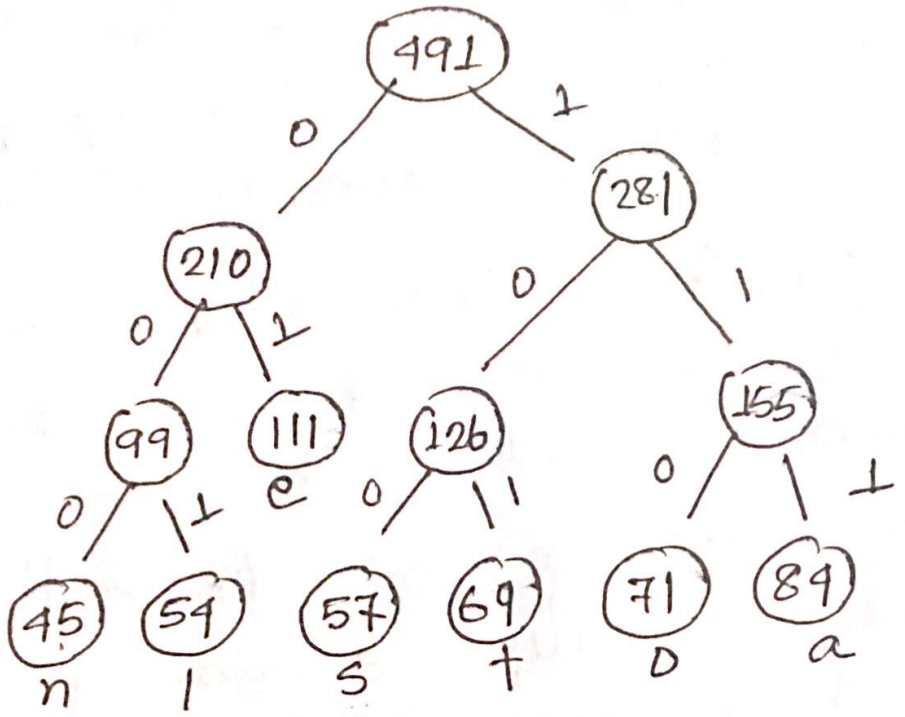
N(6,8) {T6, T4}

↓

N(8,8) {T4}

result is [800, 835], [1000, 1030], [1200, 1380], [1500, 1650] [1700, 2000]

5)



char	code	Freq	Total
a	111	84	252
e	01	111	222
l	001	54	162
n	000	45	135
o	110	71	213
s	100	57	171
T	101	69	207

Σ= 1362 bits

ASCII total = 8*491

= 3928

So, saves = 3928 − 1362

= 2566

= $\frac{2566}{3928} \times 100$

= 65.33 % are the percentages of saving

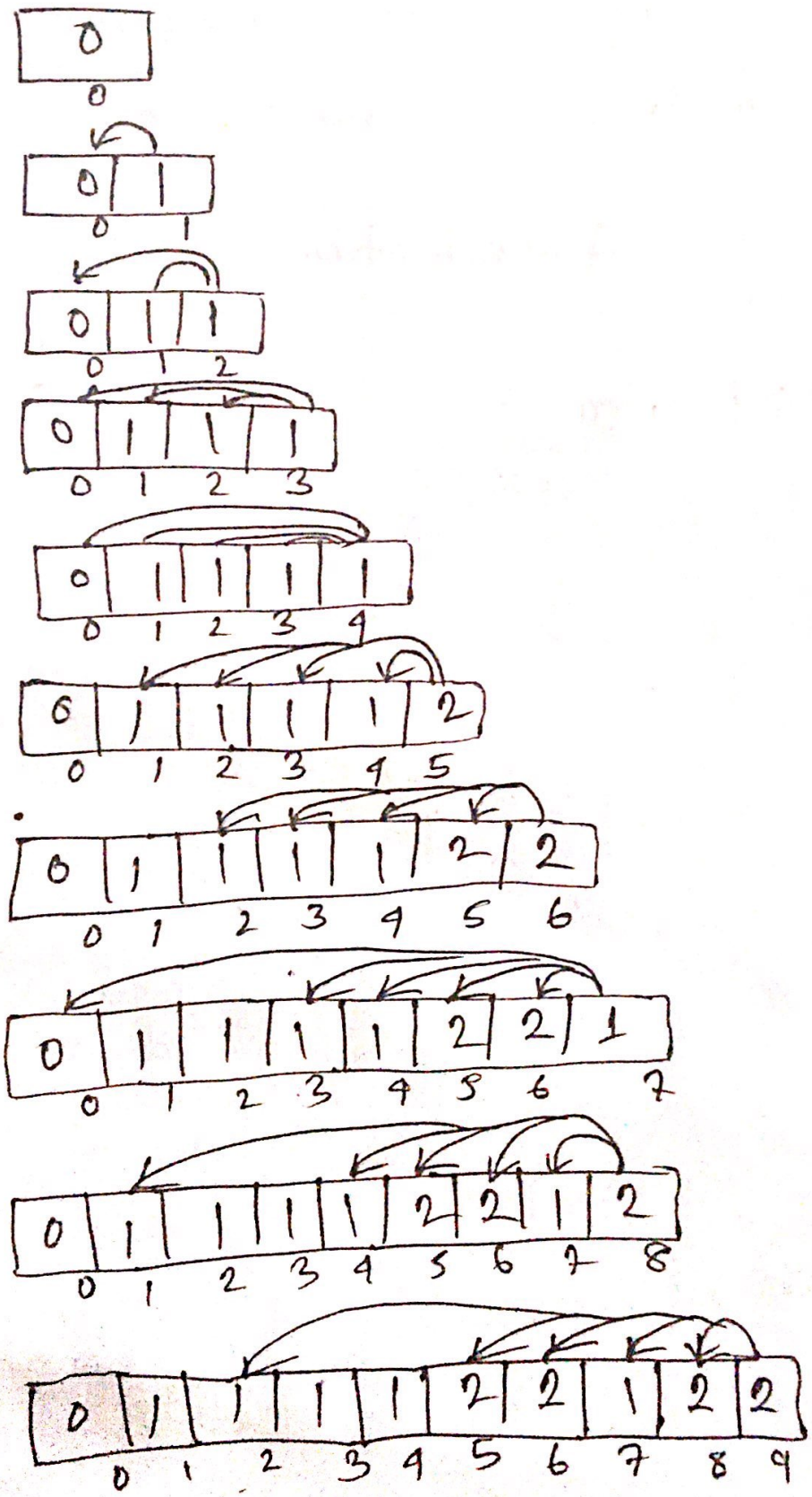
Encode 'stolen'

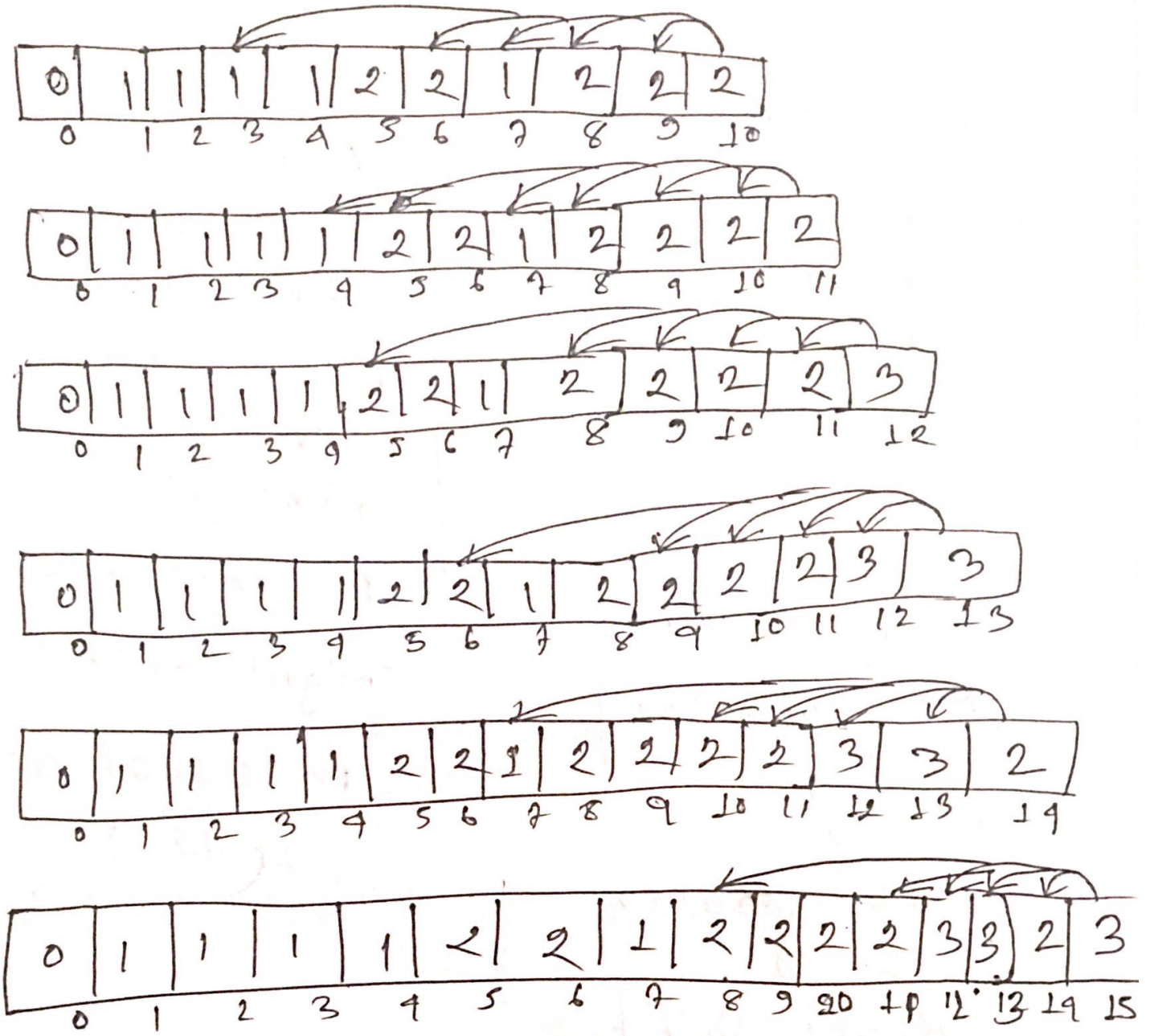
stolen = 100 101 110 001 01 000

6) Coins = {1, 2, 3, 4, 7}

Amount = 15

simulation:





minimum coin for 15 will be

(3).

7|

For example, we have a Rod that length = 8 inces

size:	1	2	3	4	5	6	7	8
values:	1	5	8	9	10	17	17	20

By using greedy approch it will select the largest value and the answer will be 20.

But, The optimal solution solution for this instance would be 6 inches and 2 inches.

$= (17+5) = 22.$

So obviously the greedy approch will be fails to find an optimal solution.

This is the optimal and greedy example.

~~8|~~ 8|

This problem is a varoiation of the knapsack problem where you're trying to maximize the total profet(revenue) while

repecting a constraint (bandwidth capacity). The dynamic programming algorithm to find the maximum total profit within the maximum capacity.

```
Max_total_p(max_capacity, band, payments)
    n = len(band)
    dp = [[0] * (max_capacity + 1)
      For : range (n+1)]
    for i range (1, n+1)
        for j range (max_capa +1)
            if band [i-1] <= j:
    dp[i] [j] = max(dp[i-1][j], dp[i-1] [
            j-band[i-1] + payments [i-1] )
        else
            dp[i][j] = dp[i-1] [j]
        return
            dp[n][max_capacity]
```

9)

(1)

values: 150 180 170 120 210

weights: 3 3 2 3 3

V / W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	(NT): 0 T: 0	(NT): 0 T: 0	(NT) 0 T: 0	(NT) 0 T: 0	(NT): 0 T: 0
2	0	(NT): 0 T: 0	(NT): 0 T 0	NT: 0 (T) 170	(NT): 170 T: 0	(NT) 170 T: 0
3	0	NT: 0 (T) 150	NT: 150 (T) 180	(NT) 180 T: 170	(NT) 180 T: 120	NT: 180 (T): 210
4	0	NT: 0 (T) 150	NT: 150 (T) 180	(NT) 180 NT: 170	(NT): 180 T: 120	NT: 180 (T) 210
5	0	NT: 0 (T) 150	NT: 150 (T) 180	NT: 180 (T) 350	(NT) 350 T: 290	NT: 350 (T) 380
6	0	NT: 0 (T) 150	NT: 150 (T) 330	NT: 330 (T) 350	(NT): 350 T: 300	NT: 350 (T) 390
7	0	NT: 0 (T) 150	NT: 150 (T) 330	NT: 330 (T) 350	(NT): 350 T: 300	NT: 350 (T) 390

Item taken:

(7,5) → Taken

(4,4)

(4,3)

(4,2) → Taken

(1,1)

(1,0)

Profit = 390.

q/ (11)

Item	1	2	3	4	5
values	3	3	2	3	3
weight	150	180	170	120	210
V/W	50	60	85	40	70

After sorting [Decending order]

Items	3	5	2	1	4
values	170	210	180	150	120
weight	2	3	3	3	3
V/W	85	70	60	50	40

step: 1

item → 3 (170)

C = (7-2) = 5

Profit = 170

step: 2

item → 5

C = (5-3) = 2

Profit = (170+210)

= 380

step: 3

item → 2

C = (2-2) = 0

(2/3 × 180) = 120

so, profeet

(170+210+120)

= 500

Algorithm (n, m)

```
for (i=1; i<=n; i++) { } ----> n
    Print(i); ----------> (n-1)
}
for(j=1; j<=m; j++) { ----> m
    for (i=1; i<=n; i++) { ---> (m-1) n
        Print(i*j); ---> (m-1)(n-1)
```

exact_cost;

$n + (n-1) + m + nm - n + (m-1)(n-1)$

$\Rightarrow$ ~~n~~ + ~~n~~ − ~~1~~ + ~~m~~ + nm − ~~n~~ + nm − ~~m~~ − ~~n~~ + ~~1~~

$\Rightarrow$ $2nm$

$\Rightarrow$ $O(nm)$

The notation;

$f(n) = 2nm$ | $g(n) = nm$

$f \leq c \cdot g(n)$

$\downarrow$ $\downarrow$

2 nm

so, $f(n) = O(nm)$

so it is big O notation.