Answer 1:

For time efficiency, we need to look into number of operations both algorithm will take by increasing value of 'n' step by step.

n	n^2	n*logn*logn
5	25	2.4
10	10^2	10
10^2	10^4	400
10^3	10^6	$<10^4$
10^4	10^8	$<10^6$
10^5	10^{10}	$<10^7$
10^6	10^{12}	$<10^8$

So, we can observe, for smaller value of n, there is not much difference in algorithms' efficiency, but the larger the value of n the bigger their efficiency difference will be. Thus, recursive algorithm is more efficient for the given problem. However, in terms of space efficiency it is not clear because of unmentioned exact problem but generally, recursive algorithms are not good in terms of space efficiency because of there stacking nature.

Answer 2:

a) Recurrence Equation:

T(n) = n + T (n/2)

b)

For all n<2:

T(0) = T1= 0 (As comparison can be made if n>1)

For all n>1:

T(2) = n + T (2/2) => (n+T(1)) => 2+0 => 2

T(4) = n + T (4/2) => (n+T(2)) => 4+2 => 6

T(8) = n + T (8/2) => (n+T(4)) => 8+6 => 14

T(16) = n + T (16/2) => (n+T(8)) => 16+14 => 30

T(32) = n + T (32/2) => (n+T(16)) => 32+30 => 62

and so on……

Following is a tree example of n=16

Tree	Steps
16	
8 8	2
4 4 4 4	4
2 2 2 2 2 2 2 2	8
11 11 11 11 11 11 11 11	16

c)

As T(n) = n + T (n/2)

So, T (n+1) = T(n/2+0.5)+n+1

= T(n+1/2)+n+1

Hence, proved

Answer 3:
We will use binary search for the given problem.
Algorithm:

```
ind_search(arr,lin,rin):
        mid = (lin + rin)/2
        if ((lin==rin) and (arr[mid] != mid))
                return “No such element”
        if (arr[mid] == mid)
                return “Found an element: “+arr[mid]
        if (arr[mid]>mid)
                ind_search(arr,lin,mid)
        if (arr[mid]<mid)
                ind_search(arr,mid,rin)
```

As we are using binary search, for every loop rin-lin will be halved so the time complexity will be O(log n)

Answer 4:
There are two solutions to solve this through divide and conquer rule
1. Binary Search Tree
2. Sorting and Extraction of most frequent element
Both the algorithm have O(n log n) time complexity.

1: Binary Search Tree (BST) Algorithm
Create binary search tree for the given array.
Declare “frequency” variable for each element/node of the array.
Whenever same variable is put in BST, increase its frequency.
Check frequency of most occurring element, if it is greater than half of size of array.

BST will take O(nlog(n))
Finding most frequent element and checking if it is greater than half of size of array will take O(n), so
Time complexity = O(nlog(n)) + O(n) => O(nlog(n))

2: Sorting and Extraction of most frequent element
Sort the array with merge sort.
Declare “count” variable, loop through the array and check if current element is same as previous one then increase the count else set count to 1.
if count is greater than half of size of array then print the element.

Divide and conquer rule is being used in merge sort. Merge sort will take O(nlog(n)) and looping through array to find the frequent element will take O(n). Thus,
Time complexity = O(nlog(n)) + O(n) => O(nlog(n))

Answer 5:

If cost of solving sub-problem increases level by level, then f(n) will be dominated by cost of last level $n^{\log_b a}$. Thus time complexity will be equal to last level complexity.

If cost of solving sub-problem at each level is nearly same then f(n) will be $n^{\log_b a}$. The complexity will be $n^{\log_b a} * \log n$.

If cost of solving sub-problem decreases level by level, then f(n) will be dominated on $n^{\log_b a}$. Thus, time complexity will be f(n).

1)

$T(n) = 9*T(n/3) + n^2 + 4$

$a = 9$

$n/b = n/3$

$f(n) = n^2+4$

$\log_b a = \log_3 9 \Rightarrow 2$

So, $n^2 < n^2+4$

As, $f(n) > n^{\log_b a}$

Thus, $T(n) = f(n) = n^2 + 4$

2)

$T(n) = 6*T(n/2) + n^2 - 2$

$a = 6$

$n/b = n/2$

$f(n) = n^2-2$

$\log_b a = \log_2 6 \Rightarrow 2.58$

So, $n^{2.58} > n^2 - 2$

As, $f(n) < n^{\log_b a}$

Thus, $T(n) = n^{2.58}$

3)

$T(n) = 4*T(n/2) + n^3 + 7$

$a=4$

$n/b=n/2$

$f(n) = n^3+7$

$\log_b a = \log_2 4 \Rightarrow 2$

So, $n^2 < n^3 + 7$

As, $f(n) > n^{\log_b a}$

Thus, $T(n) = n^3+7$

Answer 6:

$T(n)=8T(n-1)-21T(n-2)+18T(n-3)$
$T(0)=0$
$T(1)=1$
$T(2)=2$

$T(n)-8T(n-1)+21T(n-2)-18T(n-3)=0$
order: n-1-(n-3)
n-n+3 = 3
characteristic equation = $r^3 - 8r^2 + 21r - 18=0$
Solution:

$(r-3)^2 (r-2)=0$
r1=2
r2=3
r3=3

$T(n) = (C1+C2n)3^n + C32^n$ (Eq a)

$T(0) = 0$
$T(0) = (C1+C2.0)3^0+C32^0$
$T(0) = (C1)1 + C3.1$
$T(0) = C1+C3 = 0$ Eq1

$T(1) = 1$
$T(1) = (C1+C2.1)3^1+C32^1$
$T(1) = (C1+C2)3 + C3.2$
$T(1) = 3C1+3C2+2C3 =1$ Eq 2

$T(2) = 2$
$T(2) = (C1+C2.2)3^2+C32^2$
$T(2) = (C1+2C2)9 + C3.4$
$T(2) = 9C1+18C2+4C3 =2$ Eq 3

C1=-C3 (Eq4 from Eq 1)

put Eq 4 in Eq2
3(-C3)+3C2+2C3=1
-3C3+3C2+2C3=1
3C2=1+C3
C2=(1+C3)/3 Eq 5

put Eq 4 and Eq 5 in Eq3
9(-C3)+18((1+C3)/2)+4C3=2
-9C3+9+9C3+4C3=2
4C3=-7
C3 = -7/4
C1 = -(-7/4) => 7/4
C2 = (1+(-7/4))/3 => (1-7/4)/3 => -3/12 => -1/4

put value of C1, C2 and C3 in Eq a
$T(n) = (C1+C2n)3^n + C32^n$
$T(n) = (7/4-1/4n)3^n + (-7/4)2^n$

Answer 7

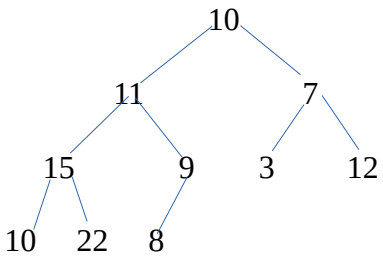
Heap
0=10
1=20
2=25
3=30
4=70
5=29
6=60=p

Delete the minimum element
As it is minHeap, minimum element is located at root level.
Replace the root with last element.
Delete the last element
Update the size of array
Now,
0=60=p
1=20
2=25
3=30
4=70
5=29
But Now 60 is at top and it is minHeap, it will not follow heap property (childrens > parent) so we will have to heapify it. After heapification:
0=20
1=30
2=25
3=60=p
4=70
5=29

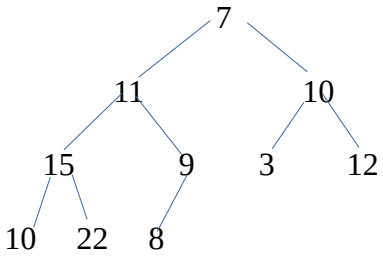
Now insert(5) in above heap
0=20
1=30
2=25
3=60=p
4=70
5=29
6=5
After heapification:
0=5
1=30
2=20
3=60=p
4=70
5=29
6=25

Answer 8:
array = [10, 11, 7, 15, 9, 3, 12, 10, 22, 8]

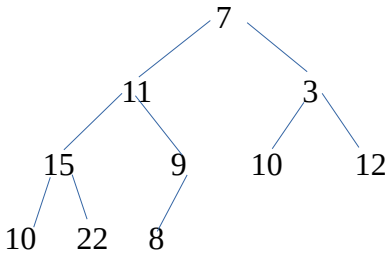
part a)
Complete Binary Tree:



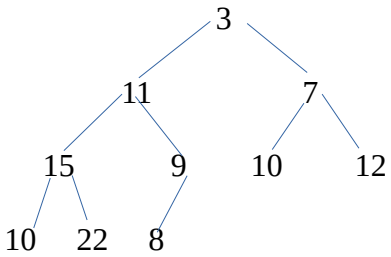
part b)
Step a: siftDown(10)
Now, 10 is first parent which is greater than its child so we will replace 10 with its child 7.



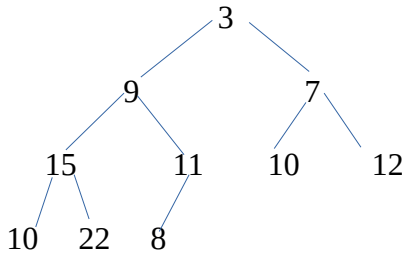
But still 10 has child (3) which is smaller than itself so we will siftDown(10) again.



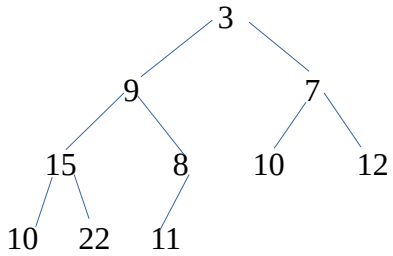
steb b: siftDown(7)
As root node is still unsatisfied with heap property, 7 is greater than its child 3 so we will sift down 7.



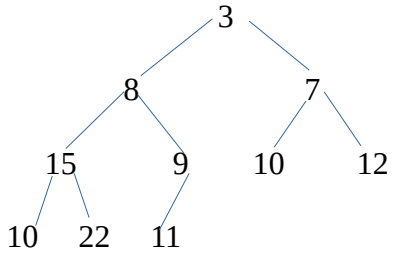
Step c: siftDown(11)
11 is greater than its child 9 so we will replace 11 with its child 9.



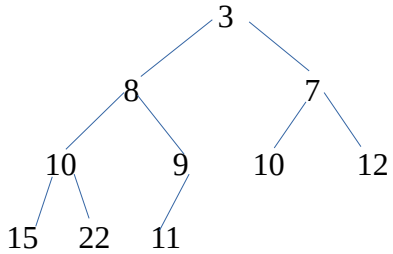
But 11 is still greater than its child 8 so we will siftDown 11 again.



Step d: siftDown(9)
Now 9 is at the current node and it is not satisfying the minheap property. We will replace 9 with its smaller child 8.

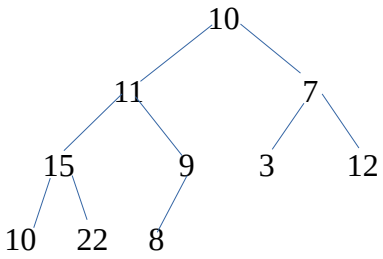


Step e: siftDown(15)
As 15 is greater than its child 10, so we will siftdown(15) with 10.



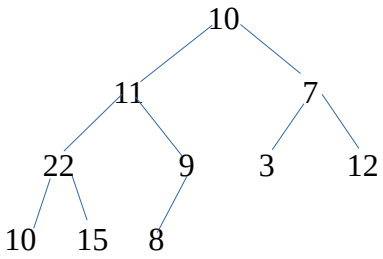
Now, every node satisfies the minHeap property.

Part c)
array = [10, 11, 7, 15, 9, 3, 12, 10, 22, 8]

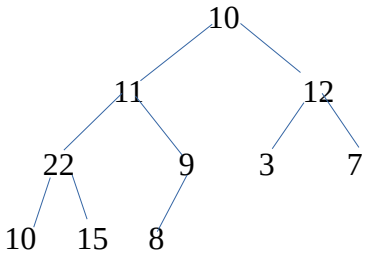


Step 1:
We will start from down to up, as 8 is less than 9 so it is ok.

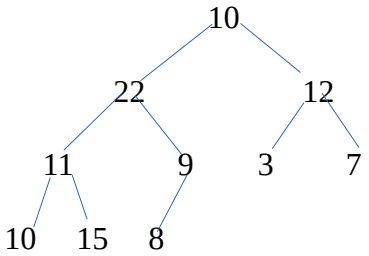
Step 2:
22 is greater than 15 so we will replace 15 with 22.



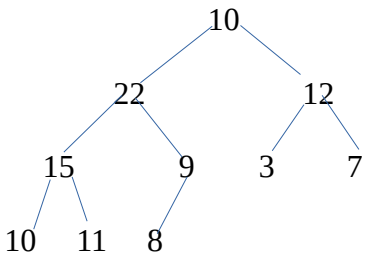
Step 3:
12 is greater than 7, so replace 7 with 12



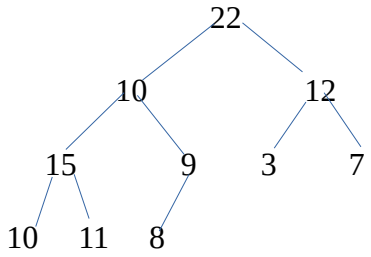
Step 4:
22 is greater than its parent 11. so replace 11 with 22.



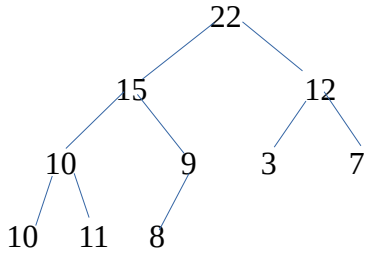
But 11 is not satisfying maxHeap property at the given position, its child 15 is greater than itself so we will replace 15 with 11.



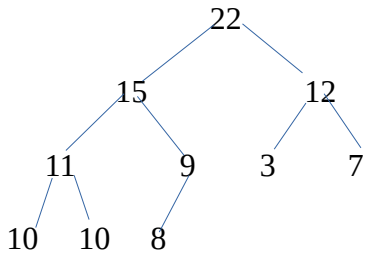
Step 5:
Root node 10 is smaller than both of its child so we will replace 10 with greater child which is 22.



But now 10 is not satisfying maxHeap property at the current position. We will replace 10 with 15.



10 is still not satisfying maxHeap property. We will replace 10 with 11.



Now, all the nodes/elements follow maxHeap property.

Answer 9:

x= CACMYCCA

y=YMCMAMYYCMA

LCS	#	C	A	C	M	Y	C	C	A
#	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	1	1	1	1
M	0	0	0	0	1	1	1	1	1
C	0	1	1	1	1	1	2	2	2
M	0	1	1	1	2	2	2	2	2
A	0	1	2	2	2	2	2	2	3
M	0	1	2	2	3	3	3	3	3
Y	0	1	2	2	3	4	4	4	4
Y	0	1	2	2	3	4	4	4	4
C	0	1	2	3	3	4	5	5	5
M	0	1	2	3	4	4	5	5	5
A	0	1	2	3	4	4	5	5	6

longest common sequence= CAMYCA

Answer 10:

For k=0

Weight matrix D0=

	1	2	3	4	5
1	0	2	inf	1	8
2	6	0	3	2	inf
3	inf	inf	0	4	inf
4	inf	inf	2	0	3
5	3	inf	inf	inf	0

path matrix P0=

	1	2	3	4	5
1	Nil	1	Nil	1	1
2	2	Nil	2	2	Nil
3	Nil	Nil	Nil	3	Nil
4	Nil	Nil	4	Nil	4
5	5	Nil	Nil	Nil	Nil

For k=1

Weight matrix D1=

	1	2	3	4	5
1	0	2	inf	1	8
2	6	0	3	2	14
3	inf	inf	0	4	inf
4	inf	inf	2	0	3
5	3	5	inf	4	0

path matrix P1=

	1	2	3	4	5
1	Nil	1	Nil	1	1
2	2	Nil	2	2	1
3	Nil	Nil	Nil	3	Nil
4	Nil	Nil	4	Nil	4
5	5	1	Nil	1	Nil

For k=2

Weight matrix D2=

	1	2	3	4	5
1	0	2	5	1	8
2	6	0	3	2	14
3	inf	inf	0	4	inf
4	inf	inf	2	0	3
5	3	5	8	4	0

path matrix P2=

	1	2	3	4	5
1	Nil	1	2	1	1
2	2	Nil	2	2	1
3	Nil	Nil	Nil	3	Nil
4	Nil	Nil	4	Nil	4
5	5	1	2	1	Nil

For k=3

Weight matrix D3=

	1	2	3	4	5
1	0	2	5	1	8
2	6	0	3	2	14
3	inf	inf	0	4	inf
4	inf	inf	2	0	3
5	3	5	8	4	0

path matrix P3=

	1	2	3	4	5
1	Nil	1	2	1	1
2	2	Nil	2	2	1
3	Nil	Nil	Nil	3	Nil
4	Nil	Nil	4	Nil	4
5	5	1	2	1	Nil

For k=4

Weight matrix D4=

	1	2	3	4	5
1	0	2	3	1	4
2	6	0	3	2	5
3	inf	inf	0	4	7
4	inf	inf	2	0	3
5	3	5	6	4	0

path matrix P4=

	1	2	3	4	5
1	Nil	1	4	1	4
2	2	Nil	2	2	4
3	Nil	Nil	Nil	3	4
4	Nil	Nil	4	Nil	4
5	5	1	4	1	Nil

For k=5

Weight matrix D5=

	1	2	3	4	5
1	0	2	3	1	4
2	6	0	3	2	5
3	10	12	0	4	7
4	6	8	2	0	3
5	3	5	6	4	0

path matrix P5=

	1	2	3	4	5
1	Nil	1	4	1	4
2	2	Nil	2	2	4
3	5	1	Nil	3	4
4	5	1	4	Nil	4
5	5	1	4	1	Nil

So D5 is our final cost matrix for each pair of elements and P5 is our final path matrix for each pair of elements.

Answer 11:

Part a)
Recurrence relation:
DP[i][j] = DP[i-1][j] or DP[i-1][j-A[i-1]]

T(x) = O(n.m)+O(n)+O(n+M)
T(x) = O(nm)

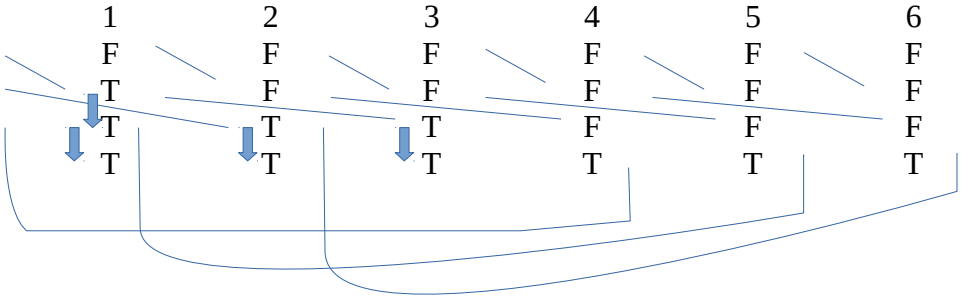
Part b)
Dynamic programming algorithm

```
is_subset_sum(set, total_sum):
        subset= []
        n=len(set)
        for i in range(n+1):
                for j in range(total_sum+1):
                        subset[i].append(False)
        for i in range(n+1):
                subset[i][0]=True
        for i in range(1,n+1):
                for j in range(1,total_sum+1):
                        if j<set[i-1]:
                                subset[i][j]=subset[i-1][j]
                        if j>=set[i-1]:
                                subset[i][j]=subset[i-1][j] or subset[i-1][j-set[i-1]]

        return subset[n][total_sum]
```

Part c)
set=[1,2,4]
sum=6

	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
1	T	T	F	F	F	F	F
2	T	T	T	T	F	F	F
4	T	T	T	T	T	T	T



So, subset[n][total_sum] = T (True) (4+2=6)

Programming Part:
There is separate folder for the programming part which has code file (.ipynb and .py), test input data result files, ‘inputfile.txt’ and readme file. Read ‘readme’ file to run the code.