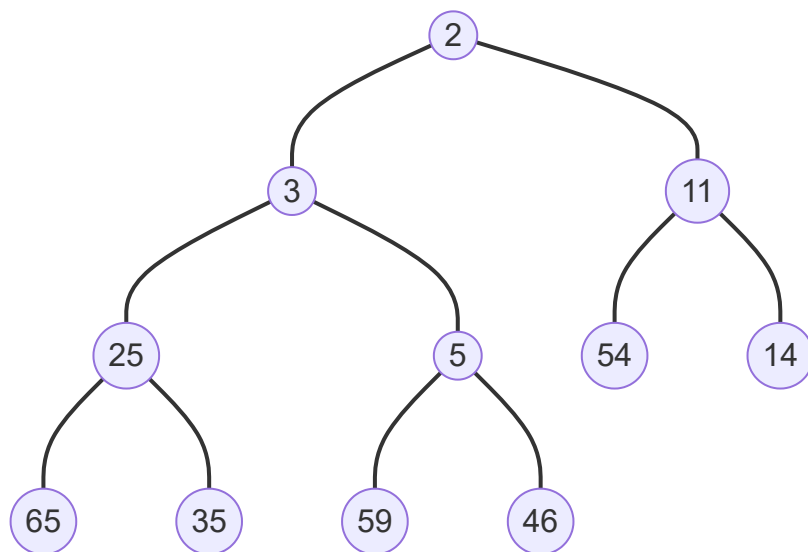


problem1

a heapsort

start:

1	2	3	4	5	6	7	8	9	10	11	12
2	3	11	25	5	54	14	65	35	59	46	



###

1	2	3	4	5	6	7	8	9	10	11	12

delete2

1	2	3	4	5	6	7	8	9	10	11	12
3	5	11	25	46	54	14	65	35	59		

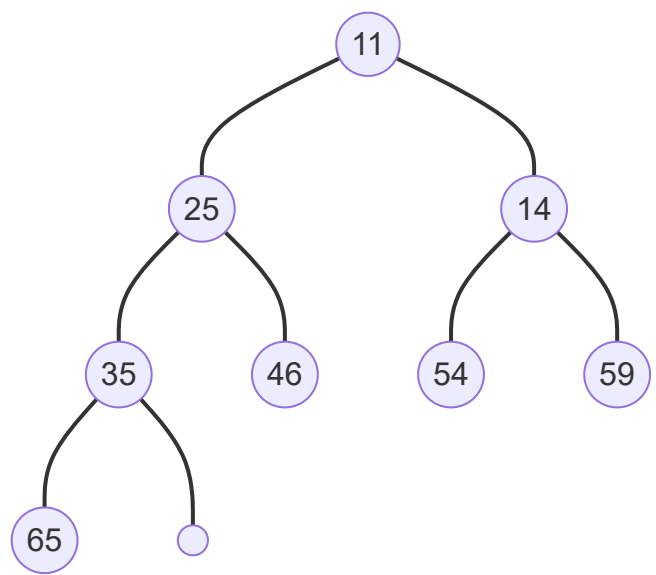


1	2	3	4	5	6	7	8	9	10	11	12
5	25	11	35	46	54	14	65	59			



delete5

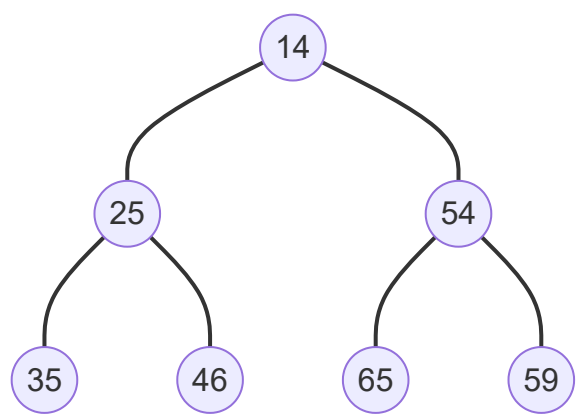
1	2	3	4	5	6	7	8	9	10	11	12
11	25	14	35	46	54	59	65				



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5									

delete11

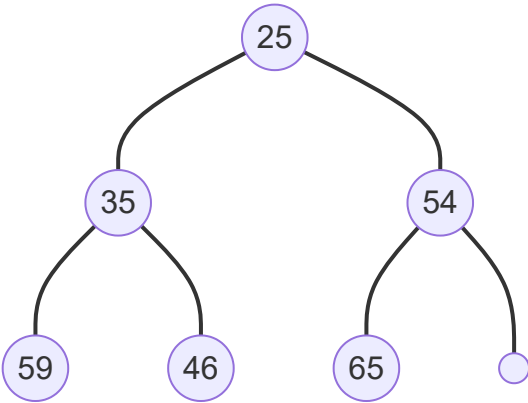
1	2	3	4	5	6	7	8	9	10	11	12
14	25	54	35	46	65	59					



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11								

delete14

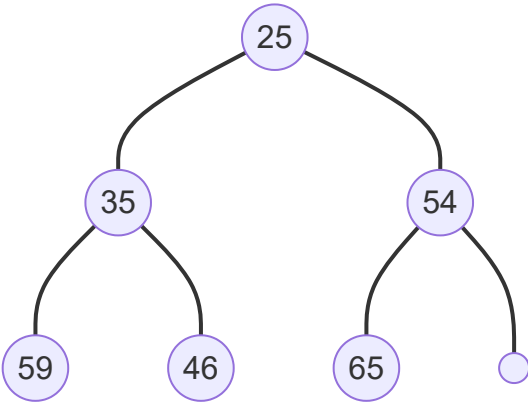
1	2	3	4	5	6	7	8	9	10	11	12
25	35	54	59	46	65						



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11	14							

delete14

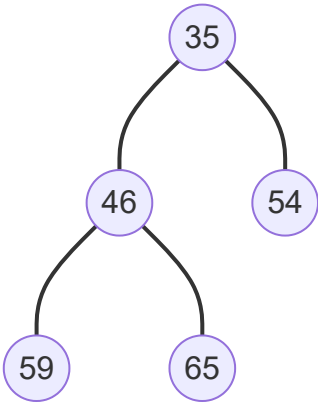
1	2	3	4	5	6	7	8	9	10	11	12
25	35	54	59	46	65						



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11	14							

delete25

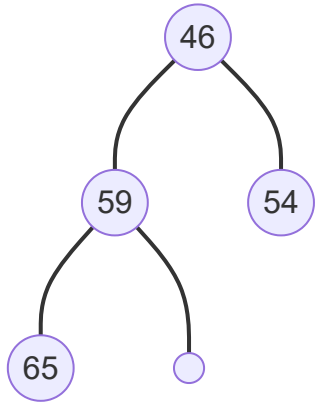
1	2	3	4	5	6	7	8	9	10	11	12
35	46	54	59	65							



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11	14	25						

delete35

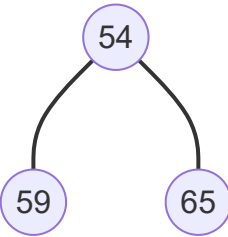
1	2	3	4	5	6	7	8	9	10	11	12
46	59	54	65								



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11	14	25	35					

delete46

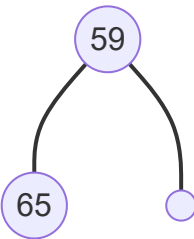
1	2	3	4	5	6	7	8	9	10	11	12
54	59	65									



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11	14	25	35	46				

delete54

1	2	3	4	5	6	7	8	9	10	11	12
59	65										



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11	14	25	35	46	54			

delete59

1	2	3	4	5	6	7	8	9	10	11	12
65											



1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11	14	25	35	46	54	59		

delete65

1	2	3	4	5	6	7	8	9	10	11	12

1	2	3	4	5	6	7	8	9	10	11	12
2	3	5	11	14	25	35	46	54	59	65	

heapsort finished

b quicksort

1	2	3	4	5	6	7	8	9	10	11	12
25	11	54	35	46	5	14	65	2	59	3	

use 25 to partition

1	2	3	4	5	6(partition)	7	8	9	10	11	12
5	11	3	2	14	25	46	65	35	59	54	

use 5,46 to partition

1	2	3(partition)	4	5	6(partition)	7	8(partition)	9	10	11	12
3	2	5	11	14	25	35	46	65	59	54	

use 3, 11, 65 to partition

1	2(partition)	3(partition)	4(partition)	5	6(partition)	7	8(partition)	9	10	11(partition)	12
2	3	5	11	14	25	35	46	54	59	65	

now all item is returned, sort finished

c mergesort

1	2	3	4	5	6	7	8	9	10	11
25	11	54	35	46	5	14	65	2	59	3

Split

1	2	3	4	5	6		7	8	9	10	11
25	11	54	35	46	5		14	65	2	59	3

Split

1	2	3		4	5	6		7	8	9		10	11
25	11	54		35	46	5		14	65	2		59	3

Split

1	2		3		4	5		6		7	8		9		10	11
25	11		54		35	46		5		14	65		2		59	3

Split

1		2		3		4		5		6		7		8		9		10		11
25		11		54		35		46		5		14		65		2		59		3

Merge

1	2		3		4	5		6		7	8		9		10		11
11	25		54		35	46		5		14	65		2		59		3

Merge

1	2	3		4	5	6		7	8	9		10	11
11	25	54		5	35	46		2	14	65		3	59

Merge

1	2	3	4	5	6		7	8	9	10	11
5	11	25	35	46	54		2	3	14	59	65

Merge

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	25	35	46	54	59	65

d insertion sort

insert 25

1	2	3	4	5	6	7	8	9	10	11
25										

insert 11

1	2	3	4	5	6	7	8	9	10	11
11	25									

insert 54

1	2	3	4	5	6	7	8	9	10	11
11	25	54								

insert 35

1	2	3	4	5	6	7	8	9	10	11
11	25	35	54							

insert 46

1	2	3	4	5	6	7	8	9	10	11
11	25	35	46	54						

insert 5

1	2	3	4	5	6	7	8	9	10	11
5	11	25	35	46	54					

insert 14

1	2	3	4	5	6	7	8	9	10	11
5	11	14	25	35	46	54				

insert 65

1	2	3	4	5	6	7	8	9	10	11
5	11	14	25	35	46	54	65			

insert 2

1	2	3	4	5	6	7	8	9	10	11
2	5	11	14	25	35	46	54	65		

insert 59

1	2	3	4	5	6	7	8	9	10	11
2	5	11	14	25	35	46	54	59	65	

insert 3

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	25	35	46	54	59	65

Insertion sort completed

e selection sort

1	2	3	4	5	6	7	8	9	10	11
25	11	54	35	46	5	14	65	2	59	3

min->2

1	2	3	4	5	6	7	8	9	10	11
2	11	54	35	46	5	14	65	25	59	3

min->3

1	2	3	4	5	6	7	8	9	10	11
2	3	54	35	46	5	14	65	25	59	11

min->5

1	2	3	4	5	6	7	8	9	10	11
2	3	5	35	46	54	14	65	25	59	11

min->11

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	46	54	14	65	25	59	35

min->14

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	54	46	65	25	59	35

min->25

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	25	46	65	54	59	35

min->35

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	25	35	65	54	59	46

min->46

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	25	35	46	54	59	65

min->54

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	25	35	46	54	59	65

min->59

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	25	35	46	54	59	65

min->65

1	2	3	4	5	6	7	8	9	10	11
2	3	5	11	14	25	35	46	54	59	65

selection sort finished

problem2

a

Minimum prefix

```
1 struct recursiveResult
2 begin
3     float leftMin;
4     int leftMinMarker;
5     float leftMax;
6     int leftMaxMarker;
7     float total;
8 end recursiveResult
9
10 function getMinimumPrefix(x[1:n])
11 begin
12     recursiveResult result;
13     if (n==1) then
14         result.leftMin = x[0];
15         result.leftMinMarker = 1;
16         result.rightMin = x[0];
17         result.rightMinMarker = 1;
18         result.total = x[0];
19         return (result);
20     endif
21     recursiveResult leftResult;
22     recursiveResult rightResult;
23     int mid;
24     mid = floor(n/2);
25     //leftside
26     leftResult = getMinimumPrefix(x[1:mid])
27     //rightside
28     rightResult = getMinimumPrefix(x[mid+1:n])
29     //get the min prefix in the three possible prefix
30     result.leftMin = min(leftResult.total *
        rightResult.leftMax, leftResult.total *
        rightResult.leftMin, leftResult.leftMin)
```

```

31 //choose accordingly to previous result
32 result.leftMinMarker =
minMarker(right.leftMaxMarker+mid,right.leftMinMarker+mid,left.leftMinMarker
)
33 //get the max prefix in the three possible prefix
34 result.leftMax = max(leftResult.total *
rightResult.leftMax,leftResult.total *
rightResult.leftMin,leftResult.leftMax)
35 //choose accordingly to previous result
36 result.leftMaxMarker =
maxMarker(right.leftMaxMarker+mid,right.leftMinMarker+mid,left.leftMaxMarker
)
37 //add the total for outer level usage
38 result.total = leftResult.total * rightResult.total;
39 return(result)
40 end getMinimumPrefix
41
42 function main()
43 begin
44     float x[1:n]=[0.1,0.2,0.3,.....]
45     int k;
46     recursiveResult result;
47     recursiveResult = getMinimumPrefix(x);
48     k = recursiveResult.leftMinMarker;
49     print(k);
50 end main

```

Minimum suffix

```

1 struct recursiveResult
2 begin
3     float rightMin;
4     int rightMinMarker;
5     float rightMax;
6     int rightMaxMarker;
7     float total;
8 end recursiveResult
9
10 function getMinimumSuffix(x[1:n])
11 begin
12     recursiveResult result;
13     if (n==1) then
14         result.rightMin = x[0];
15         result.rightMinMarker = 1;
16         result.rightMax = x[0];
17         result.rightMaxMarker = 1;
18         result.total = x[0];
19         return (result);
20     endif
21     recursiveResult leftResult;
22     recursiveResult rightResult;
23     int mid;
24     mid = floor(n/2);
25     //leftside
26     leftResult = getMinimumSuffix(x[1:mid])
27     //rightside
28     rightResult = getMinimumSuffix(x[mid+1:n])

```

```

29 //get the min suffix in the three possible suffix
30 result.leftMin = min(rightResult.total *
leftResult.rightMax, rightResult.total *
leftResult.rightMin, rightResult.rightMin)
31 //choose accordingly to previous result
32 result.rightMinMarker =
minMarker(left.rightMaxMarker, left.rightMinMarker, right.rightMinMarker+mid)
33 //get the max suffix in the three possible suffix
34 result.leftMax = max(rightResult.total *
leftResult.rightMax, rightResult.total *
leftResult.rightMin, rightResult.rightMax)
35 //choose accordingly to previous result
36 result.leftMaxMarker =
maxMarker(left.rightMaxMarker, left.rightMinMarker, right.rightMaxMarker+mid)
37 //add the total for outer level usage
38 result.total = leftResult.total * rightResult.total;
39 return(result)
40 end getMinimumPrefix
41
42 function main()
43 begin
44     float x[1:n]=[0.1,0.2,0.3,.....]
45     int k;
46     recursiveResult result;
47     recursiveResult = getMinimumSuffix(x);
48     k = recursiveResult.rightMinMarker;
49     print(k);
50 end main

```

Minimum subarray

```

1 struct recursiveResult
2 begin
3     float rightMin;
4     int rightMinMarker;
5     float rightMax;
6     int rightMaxMarker;
7     float leftMin;
8     int leftMinMarker;
9     float leftMax;
10    int leftMaxMarker;
11    float middleMin;
12    int middleMinLeftMarker;
13    int middleMinRightMarker;
14    float total;
15 end recursiveResult
16
17 function getMinimumSubarray(x[1:n])
18 begin
19     recursiveResult result;
20     if (n==1) then
21         result.rightMin = x[0];
22         result.rightMinMarker = 1;
23         result.leftMin = x[0];
24         result.leftMinMarker = 1;
25         result.middleMin = x[0];
26         result.middleMinLeftMarker = 1;

```

```

27     result.middleMinRightMarker = 1;
28     result.total = x[0];
29     return (result);
30 endif
31 recursiveResult leftResult;
32 recursiveResult rightResult;
33 int mid;
34 mid = floor(n/2);
35 //leftside
36 leftResult = getMinimumSubarray(x[1:mid])
37 //rightside
38 rightResult = getMinimumSubarray(x[mid+1:n])
39
40 //get the min prefix in the three possible prefix
41 result.leftMin = min(leftResult.total *
rightResult.leftMax, leftResult.total *
rightResult.leftMin, leftResult.leftMin)
42 //choose accordingly to previous result
43 result.leftMinMarker =
minMarker(right.leftMaxMarker+mid, right.leftMinMarker+mid, left.leftMinMarker
)
44 //get the max prefix in the three possible prefix
45 result.leftMax = max(leftResult.total *
rightResult.leftMax, leftResult.total *
rightResult.leftMin, leftResult.leftMax)
46 //choose accordingly to previous result
47 result.leftMaxMarker =
maxMarker(right.leftMaxMarker+mid, right.leftMinMarker+mid, left.leftMaxMarker
)
48
49 //get the min suffix in the three possible suffix
50 result.leftMin = min(rightResult.total *
leftResult.rightMax, rightResult.total *
leftResult.rightMin, rightResult.rightMin)
51 //choose accordingly to previous result
52 result.rightMinMarker =
minMarker(left.rightMaxMarker, left.rightMinMarker, right.rightMinMarker+mid)
53 //get the max suffix in the three possible suffix
54 result.leftMax = max(rightResult.total *
leftResult.rightMax, rightResult.total *
leftResult.rightMin, rightResult.rightMax)
55 //choose accordingly to previous result
56 result.leftMaxMarker =
maxMarker(left.rightMaxMarker, left.rightMinMarker, right.rightMaxMarker+mid)
57
58 //get the min suffix in the 6 possible min suffixs
59 result.middleMin =
min(leftResult.middleMin, rightResult.middleMin, leftResult.rightMin*rightResu
lt.leftMin,
leftResult.rightMin*rightResult.leftMax, leftResult.rightMax*rightResult.lef
tMin, leftResult.rightMax*rightResult.leftMax)
60 //get the left marker accordingly
61 result.middleMinLeftMarker =
minLeftMarker(leftResult.middleMinLeftMarker, rightResult.middleMinLeftMarker
+mid, leftResult.rightMinMarker, leftResult.rightMinMarker, leftResult.rightMax
Marker, leftResult.rightMaxMarker)
62 //get the right marker accordingly

```

```

63     result.middleMinRightMarker =
minRightMarker(leftResult.middleMinRightMarker, rightResult.middleMinRightMar
ker+mid, rightResult.leftMinMarker+mid, rightResult.leftMaxMarker+mid, rightRes
ult.leftMinMarker+mid, rightResult.leftMaxMarker+mid)
64
65     //add the total for outer level usage
66     result.total = leftResult.total * rightResult.total;
67     return(result);
68 end getMinimumSubarray
69
70 function main()
71 begin
72     float x[1:n]=[0.1,0.2,0.3,.....]
73     int k,r;
74     recursiveResult result;
75     recursiveResult = getMinimumSubarray(x);
76     k = recursiveResult.middleMinLeftMarker;
77     r = recursiveResult.middleMinRightMarker;
78     print(k,r);
79 end main

```

C

About Time Complexity, the three algorithms above are all acting in a same way, that is recursively split the input into two half and calls themselves, then each recursive call it self does constant operations. So

$$\begin{aligned}
 T(1) &= c \\
 T(n) &= T\left(\frac{n}{2}\right) + c \\
 \text{Therefore} \\
 T(n) &= O(\log n)
 \end{aligned}
 \tag{1}$$

problem3

```

1  struct recursiveResult
2  begin
3      float maxTrough;
4      int maxTroughMarker;
5  end recursiveResult
6
7  struct minHeapNode
8  begin
9      float value;
10     int location;
11 end minHeapNode
12
13 function buildMinHeap(x[1:n])
14 begin
15     int i;
16     minHeapNode minHeap[1:n];
17     int mapArray[1:n];
18     for i=1 to n do
19         //insert into min heap
20         minHeapNode heapNode;

```



```

21     heapNode.value = x[i];
22     heapNode.location = i;
23     minHeap[i] = heapNode;
24     mapArray[i] = i;
25     int currentLocation;
26     currentLocation = i;
27     while (currentLocation != 1 and minHeap[currentLocation].value <
minHeap[floor(currentLocation/2)].value) do
28         //swap both minheap and the minheap mapping array
29         swap
30         (minHeap[currentLocation], minHeap[floor(currentLocation/2)]);
31         swap
32         (mapArray[minHeap[currentLocation].location], mapArray[minHeap[floor(current
Location/2)].location]);
33         currentLocation = floor(currentLocation/2);
34     endwhile
35 endfor
36 return (minHeap, mapArray);
37 end buildMinHeap
38
39 function heapify(x[1:n], mapArray[1:n], location)
40 begin
41     while ((location < n/2) && (x[location].value > x[location*2].value
|| x[location].value > x[location*2+1].value) || ((location != 1) &&
(x[location].value < x[floor(location/2)].value)) do
42         if (x[location].value > x[location*2].value
|| x[location].value > x[location*2+1].value) then
43             //swap down
44             if x[location*2].value > x[location*2+1].value then
45                 swap (x[location], x[location*2+1]);
46             swap
47             (mapArray[x[location].location], mapArray[x[location*2+1].location]);
48             location = floor(location*2+1);
49         else
50             swap (x[location], x[location*2]);
51             swap
52             (mapArray[x[location].location], mapArray[x[location*2].location]);
53             location = floor(location*2);
54         endif
55     endwhile
56     return x, mapArray;
57 end heapify
58
59 function getTrough(x[1:n], l)
60 begin
61     int mid = floor(n/2);
62     float midTrough;
63     int midTroughMarker;
64     int i;
65     float arrayToCheck[];
66     int arrayLength;

```

```

69     if 2*l>n then
70         //the basic unit of recursive
71         arrayToCheck = x;
72         arrayLength = n;
73     else
74         //the middle unit of recursive
75         arrayToCheck = x[mid-l+1:mid+1]
76         arrayLength = 2*l;
77     endif
78     //build a l size min heap, with mapping to the element's location in a
79     //this minheap also has a location value for it to map to the array
    when doing heapify
80     //also return a l size location mapArray for each element in array x to
    find it's location in the min heap.
81     minHeapNode minHeap[1:1];
82     int mapArray[1:1];
83     minHeap,mapArray := buildMinHeap(arrayToCheck[1:1]);
84     midTroughMarker = 1;
85     midTrough = minHeap[1].value;
86     for i=l+1 to arrayLength do
87         // replace old value with new one
88         minHeap[mapArray[i%1]].value = arrayToCheck[i];
89         // rebuild the heap, return the new heap and the heap mapArray
90         minHeap,mapArray =
    heapify(arrayToCheck[1:arrayLength];mapArray[1:1],mapArray[i%1]);
91         // check for trough, the 1st element in min heap is always min
92         if midTrough<minHeap[1].value then
93             //record the new max trough
94             midTrough = minHeap[1].value;
95             midTroughMarker = i-1;
96         endif
97     endfor
98     recursiveResult result;
99     if 2*l>n then
100         //the basic unit of recursive
101         result.maxTrough = midTrough;
102         result.maxTroughMarker = midTroughMarker;
103     else
104         //start recursive here
105         recursiveResult leftResult;
106         recursiveResult rightResult;
107         leftResult = getTrough(x[1:mid],l);
108         rightResult = getTrough(x[mid+1:n],l);
109         if (midTrough >= leftResult.maxTrough && midTrough >=
    rightResult.maxTrough) then
110             result.maxTrough = midTrough;
111             //need to add offset to marker
112             result.maxTroughMarker = mid-l+midTroughMarker-1;
113         else if leftResult.maxTrough >= rightResult.maxTrough then
114             result.maxTrough = leftResult.maxTrough;
115             result.maxTroughMarker = leftResult.maxTroughMarker;
116         else
117             result.maxTrough = rightResult.maxTrough;
118             result.maxTroughMarker = rightResult.maxTroughMarker + mid;
119         endif
120     return (result);
121 end getTrough
122

```

```

123
124 func main()
125 begin
126     //populate x and l here
127     float x[1:n]=[1,2,3,4,5,6,7,8...]
128     int l = 22;
129     print(getTrough(x,l).maxTroughMarker)
130 end main

```

For Time complexity, the algorithm calls itself on half recursively. And each recursive operation will have 1 time of building min-heap($O(l)$) and l times of heapify on l elements ($\log l$ for each heapify) operations. So

$$\begin{aligned}
 T(2 * l) &= c * l \log l \\
 T(n) &= T\left(\frac{n}{2}\right) + c * l \log l \\
 \text{Therefore} \\
 T(n) &= O(l \log l \log n)
 \end{aligned}
 \tag{2}$$

problem 4

For the knapsack problem, a greedy solution on price per weight is needed

Calculate price per weight first

weight	8	14	6	4	2	10
price	40	14	24	12	12	20
price/weight	5	1	4	3	6	2
x						

M=14

firstly choose the highest price/weight which is 6

weight	8	14	6	4	2	10
price	40	14	24	12	12	20
price/weight	5	1	4	3	6	2
x					2	

M=12

choose the highest price/weight which is other than 6
(5)

weight	8	14	6	4	2	10
price	40	14	24	12	12	20
price/weight	5	1	4	3	6	2
x	8				2	

M=4

choose the highest price/weight which is other than 6, 5
(4)

weight	8	14	6	4	2	10
price	40	14	24	12	12	20
price/weight	5	1	4	3	6	2
x	1		2/3		1	

M=0

because the remaining capacity is only 4, only 4/6 of the total weight can be taken. so $x = 2/3$, and now the pack is full.

So the x_n is shown below

n	1	2	3	4	5	6
x_n	1	0	2/3	0	1	0

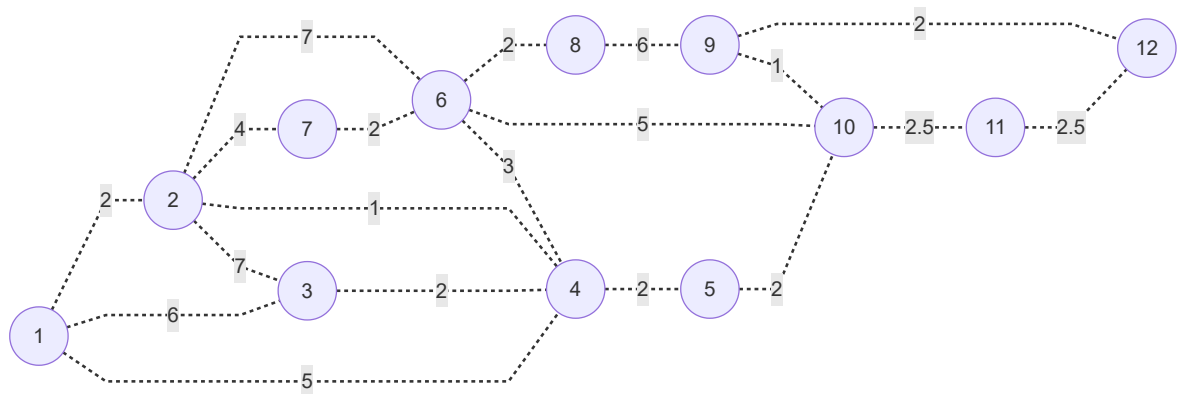
total price should be

$$x_1P_1 + x_3P_3 + x_5P_5 = 40 + 16 + 12 = 68$$

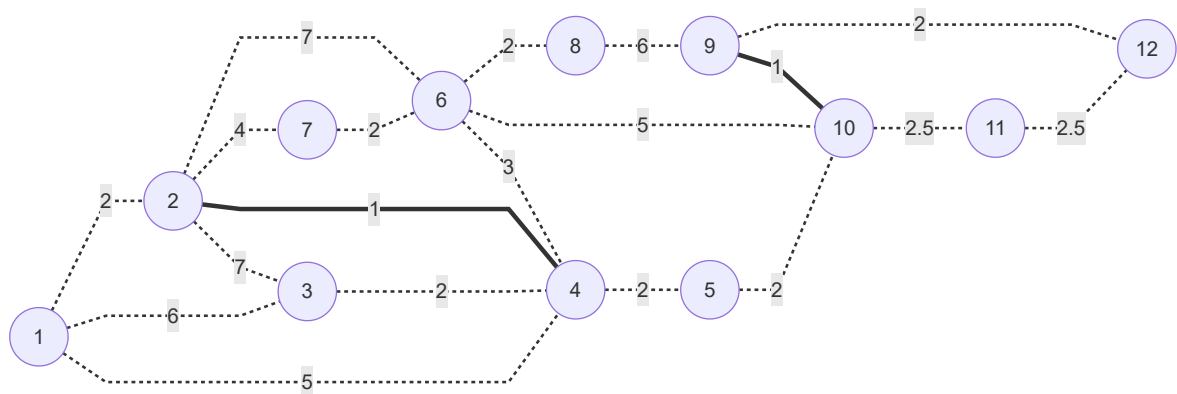
problem 5

a

We have the graph G as below



Search for the smallest weight (1), add to the tree(the bold line)



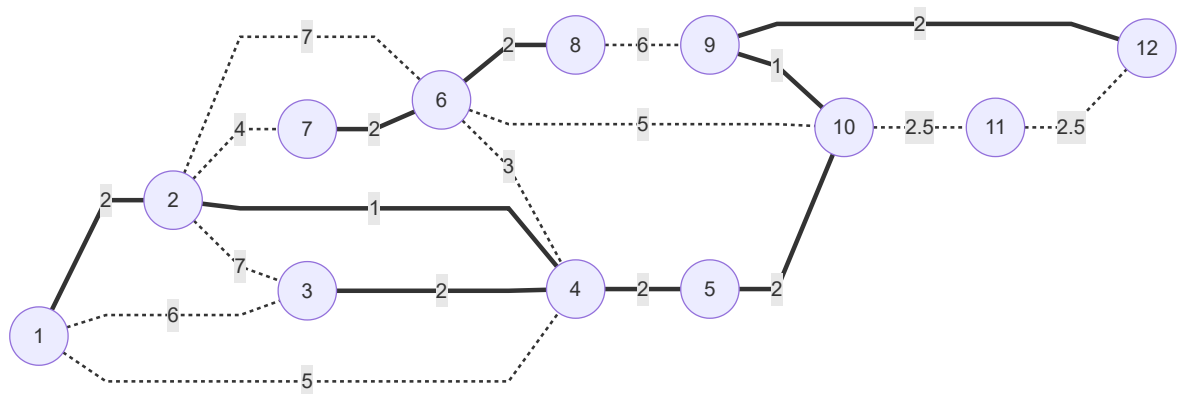
add $E(2,4), E(9,10)$

check for circle, no circle

check for $\text{count}(E)=2 \neq 11$

ok continue

Search for the smallest weight (2), add to the tree



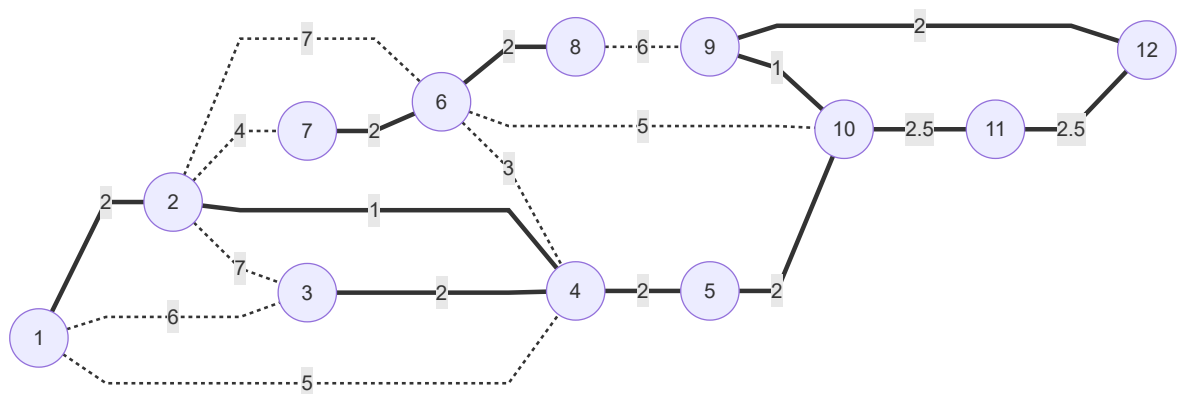
Add $E(1,2), E(3,4), E(6,7), E(4,5), E(5,10), E(9,10), E(6,8)$

check for circle, no circle

check for $\text{count}(E)=9 \neq 11$

ok continue

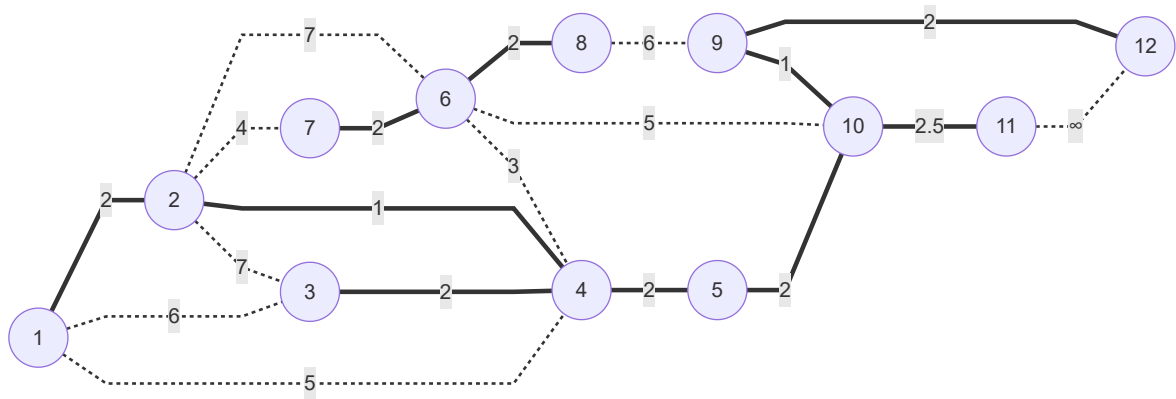
Search for the smallest weight (2.5), add to the tree



Add $E(10,11), E(11,12)$

check for circle, adding $E(11,12)$, vertex 11,12 is on the same tree, 9,10,11,12 is a circle, so remove $E(11,12)$

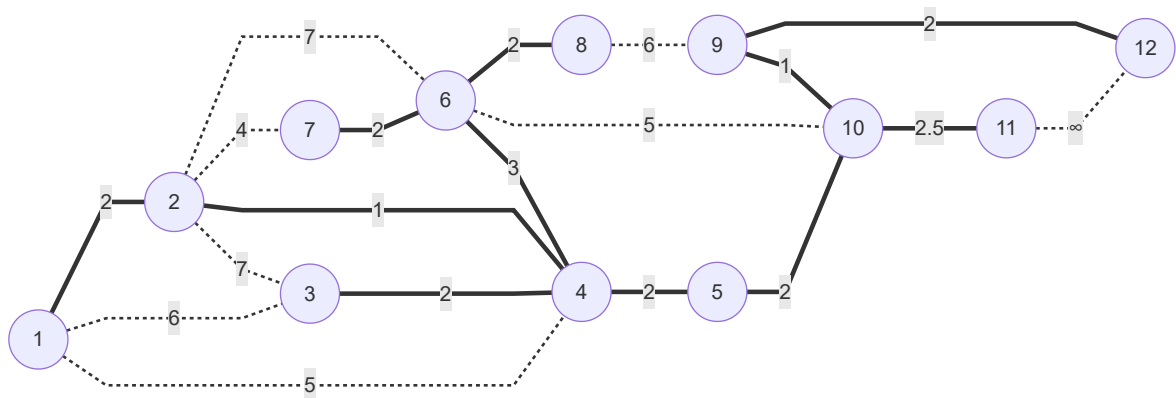
###



check for $\text{count}(E)=10 \neq 11$

ok continue

Search for the smallest weight (3), add to the tree



check for circle, adding $E(4,6)$ is on the different tree, no circle

check for $\text{count}(E)=11 == 11$

so now the tree should be a **MST** of G

b

Add 1 to group

distance from 1

i	1+	2	3	4	5	6	7	8	9	10	11	12
D[i]	0	2	6	5	∞	∞	∞	∞	∞	∞	∞	∞

choose the shortest distance (D[2]=2)to add to group

refresh distance from 1

$$D(1,2)+D(2,4)=3 > D(1,4)=5$$

i	1+	2+	3	4	5	6	7	8	9	10	11	12
D[i]	0	2	6	3	∞	9	6	∞	∞	∞	∞	∞

choose the shortest distance (D[4]=3)to add to group

refresh distance from 1

$$D(1,4)+D(4,6)=6 > D(1,6)=9$$

$$D(1,4)+D(4,3)=5 > D(1,3)=6$$

i	1+	2+	3	4+	5	6	7	8	9	10	11	12
D[i]	0	2	5	3	5	6	6	∞	∞	∞	∞	∞

choose the shortest distance (D[5]=5)to add to group

refresh distance from 1

i	1+	2+	3	4+	5+	6	7	8	9	10	11	12
D[i]	0	2	5	3	5	6	6	∞	∞	7	∞	∞

choose the shortest distance (D[3]=5)to add to group

refresh distance from 1

i	1+	2+	3+	4+	5+	6	7	8	9	10	11	12
D[i]	0	2	5	3	5	6	6	∞	∞	7	∞	∞

choose the shortest distance (D[6]=6)to add to group

refresh distance from 1

i	1+	2+	3+	4+	5+	6+	7	8	9	10	11	12
D[i]	0	2	5	3	5	6	6	8	∞	7	∞	∞

choose the shortest distance ($D[7]=6$)to add to group

refresh distance from 1

i	1+	2+	3+	4+	5+	6+	7+	8	9	10	11	12
D[i]	0	2	5	3	5	6	6	8	∞	7	∞	∞

choose the shortest distance ($D[10]=7$)to add to group

refresh distance from 1

i	1+	2+	3+	4+	5+	6+	7+	8	9	10+	11	12
D[i]	0	2	5	3	5	6	6	8	8	7	9.5	∞

choose the shortest distance ($D[8]=8$)to add to group

refresh distance from 1

i	1+	2+	3+	4+	5+	6+	7+	8+	9	10+	11	12
D[i]	0	2	5	3	5	6	6	8	8	7	9.5	∞

choose the shortest distance ($D[9]=8$)to add to group

refresh distance from 1

i	1+	2+	3+	4+	5+	6+	7+	8+	9	10+	11	12
D[i]	0	2	5	3	5	6	6	8	8	7	9.5	10

choose the shortest distance ($D[11]=9.5$)to add to group

refresh distance from 1

i	1+	2+	3+	4+	5+	6+	7+	8+	9+	10+	11+	12
D[i]	0	2	5	3	5	6	6	8	8	7	9.5	10

choose the shortest distance ($D[12]=10$)to add to group

refresh distance from 1

i	1+	2+	3+	4+	5+	6+	7+	8+	9+	10+	11+	12+
D[i]	0	2	5	3	5	6	6	8	8	7	9.5	10

now that all the elements are added to the group, finished.

C

2: 1->2

3: 1->2->4->3

4: 1->2->4

5: 1->2->4->5

6: 1->2->4->6

7: 1->2->7

8: 1->2->4->6->8

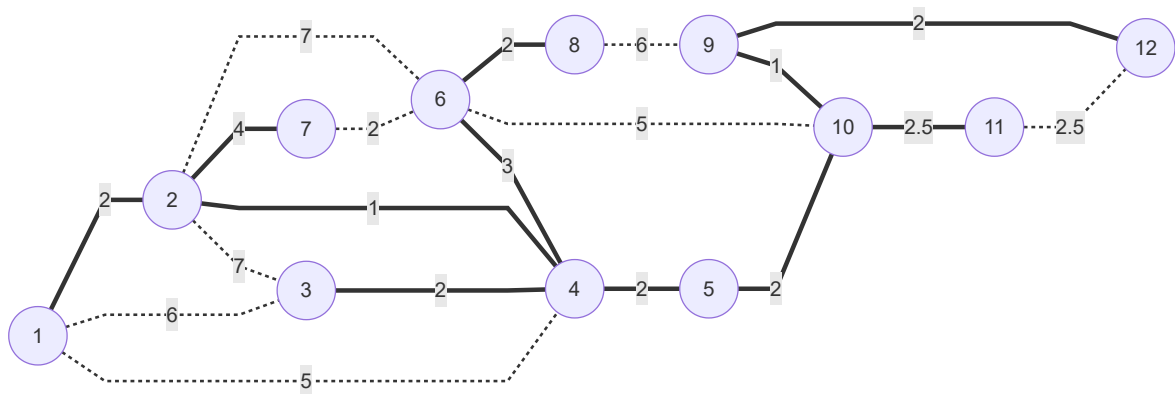
9: 1->2->4->5->10->9

10: 1->2->4->5->10

11: 1->2->4->5->10->11

12: 1->2->4->5->10->9->12

To add these paths all together, got the spanning tree below

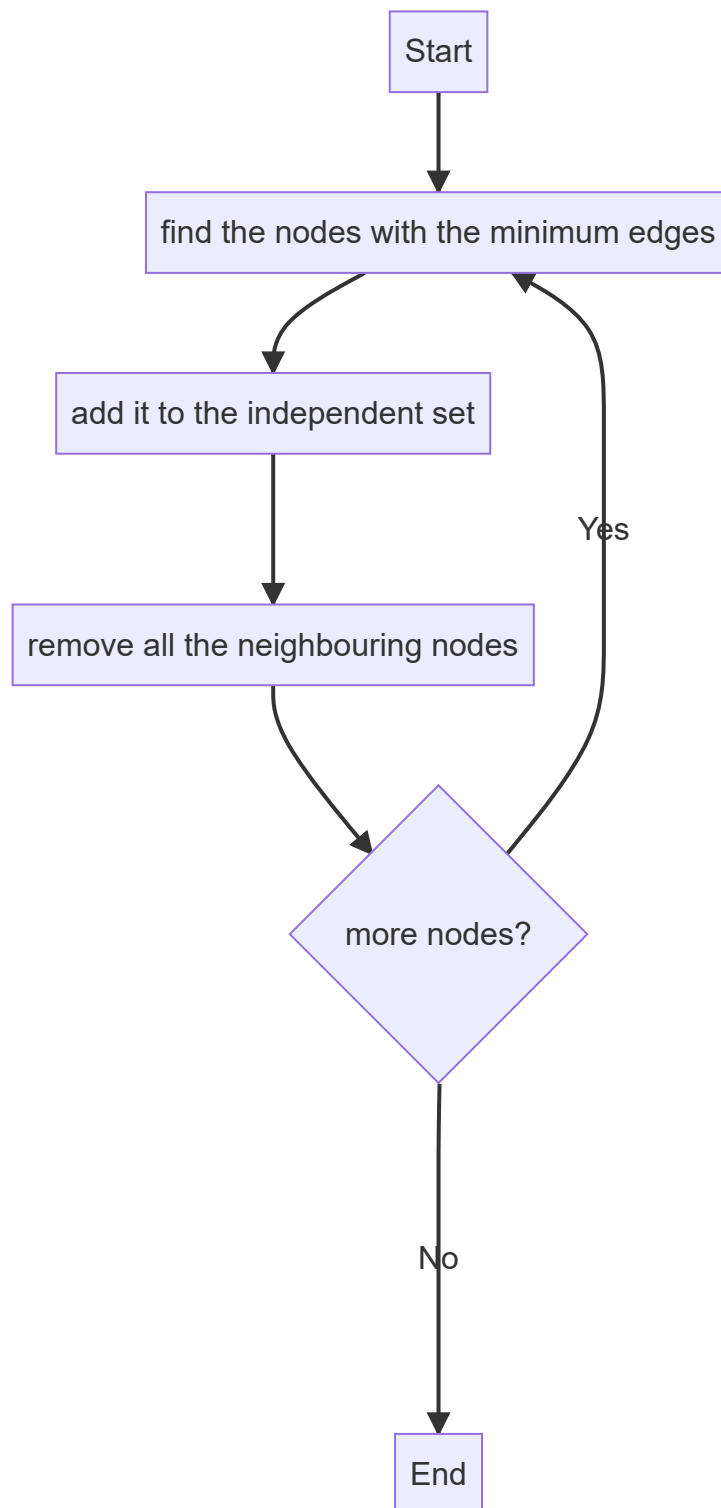


This is not a minimum spanning tree, for this tree added $E(2,7) = 4$ instead of the minimum edge $E(6,7) = 2$

problem6

a

basic idea



By removing the node with minimum neighbours one by one, hope this algorithm will have the maximum removal times, in order to bring out the maximum independent set

pesudo code

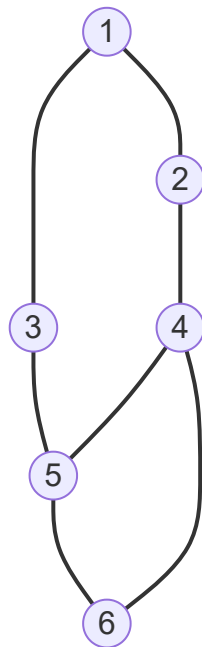
```
1 function min(A[1:n])
2 begin
3     int min = A[1];
4     for int i=2 to n
5         if min>a[i] then
6             min = a[i]
7         endif
```

```

8      endfor
9      return min
10   end min
11
12   function minLocation(A[1:n])
13   begin
14       int min = A[1];
15       int mark = 1;
16       for int i=2 to n
17           if min>a[i] then
18               min = a[i]
19               mark = i;
20           endif
21       endfor
22       return mark
23   endmin
24
25   //assume G is an adjacency matrix of n node graph
26   function greedyGetMaxIndependentSet(G[1:n][1:n])
27   begin
28       //count edge for each node
29       int edgeNum[1:n];
30       int independentSet[1:n];
31       for int i=1 to n do
32           for int j=i+1 to n do
33               if G[i][j]=1 then
34                   edgeNum[i]++;
35                   edgeNum[j]++;
36               endif
37           endfor
38       endfor
39       int counter = 0;
40       for min(edgeNum[1:n])!=0 do
41           //get node with min edges
42           int currentLocation := minLocation(edgeNum[1:n]);
43           //add to independent set
44           independentSet[counter]=currentLocation;
45           counter ++;
46           //remove neighbours
47           for int i=1 to n do
48               if G[currentLocation][i] == 1 then
49                   for int j=1 to n do
50                       if G[i][j]!=0 then
51                           edgeNum[i]--;
52                           edgeNum[j]--;
53                           G[i][j]=0;
54                           G[j][i]=0;
55                       endif
56                   endfor
57               endif
58           endfor
59           edgeNum[currentLocation]=0;
60       endfor
61       return independentSet[1:counter];
62   end greedyGetMaxIndependentSet

```

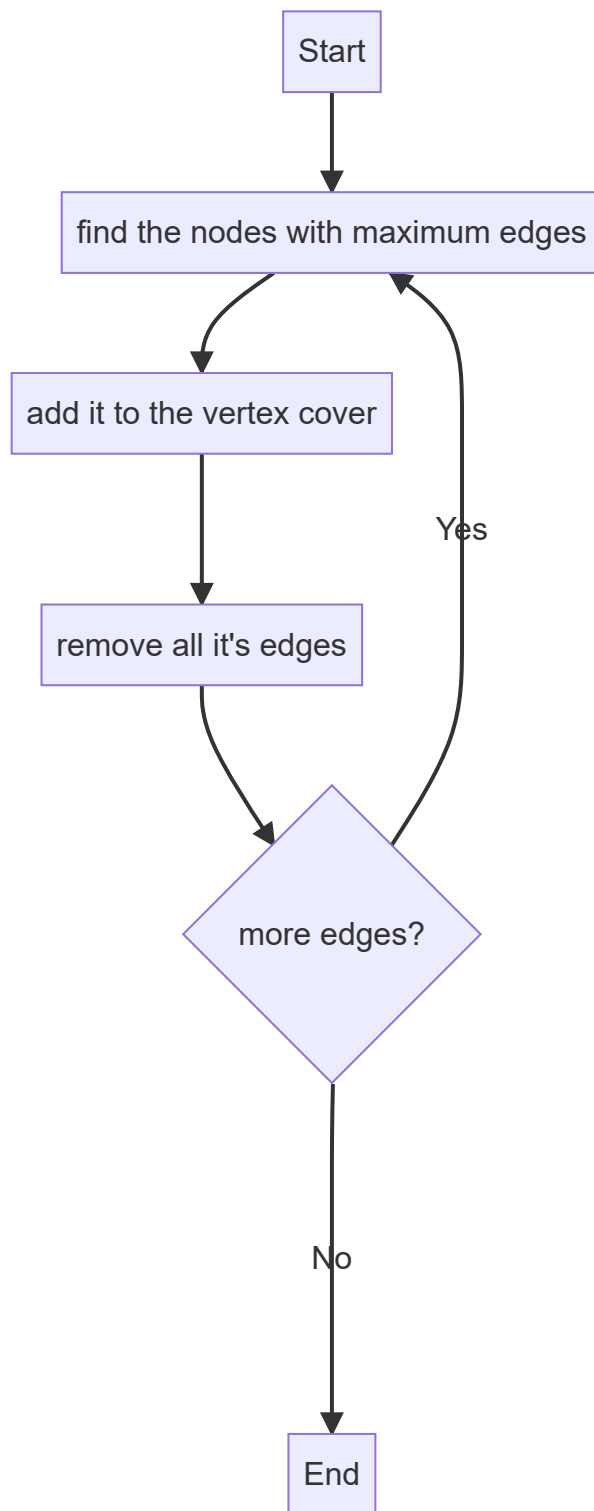
Counter Example:



In this case, 1,2,3,6 all have 2 edges, but if we select node 1 as the first node to remove, we will result in (1,4) as the maximum set, while choosing 2,3,6 at start node will have a maximum set of 3 (2,3,6). So in this case, greedy is not always optimal.

b

basic idea



For edges removed in each step is the current maximum, after removed all edges, we hope this algorithm will bring out the minimum vertex cover

pesudo code

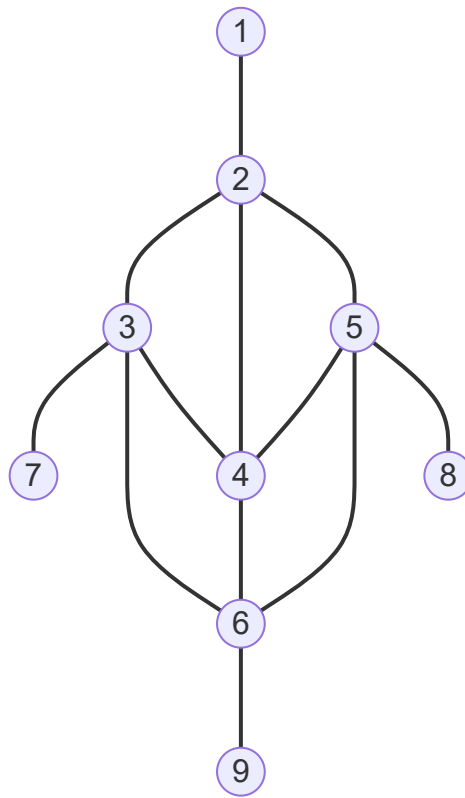
```
1 function max(A[1:n])
2 begin
3     int max = A[1];
4     for int i=2 to n
5         if max<a[i] then
6             max = a[i]
7         endif
8     endfor
```

```

9      return max
10   end min
11
12   function maxLocation(A[1:n])
13   begin
14       int max = A[1];
15       int mark = 1;
16       for int i=2 to n
17           if max<a[i] then
18               max = a[i]
19               mark = i;
20           endif
21       endfor
22       return mark
23   endmin
24
25   //assume G is an adjacency matrix of n node graph
26   function greedyGetMinVertexCover(G[1:n][1:n])
27   begin
28       //count edge for each node
29       int edgeNum[1:n];
30       int vertexCover[1:n];
31       for int i=1 to n do
32           for int j=i+1 to n do
33               if G[i][j]=1 then
34                   edgeNum[i]++;
35                   edgeNum[j]++;
36               endif
37           endfor
38       endfor
39       int counter = 0;
40       for max(edgeNum[1:n])!=0 do
41           //get node with max edges
42           int currentLocation = maxLocation(edgeNum[1:n]);
43           //add to vertex cover set
44           vertexCover[counter]=currentLocation;
45           counter ++;
46           //remove edges
47           for int i=1 to n do
48               if G[currentLocation][i] == 1 then
49                   edgeNum[i]--;
50                   G[i][currentLocation]=0;
51                   G[currentLocation][i]=0;
52               endif
53           endfor
54           edgeNum[currentLocation] = 0;
55       endfor
56       return vertexCover[1:counter];
57   end greedyGetMinVertexCover

```

Counter Example:



In this case, we can see 2,3,4,5,6 all have 4 edges, if we remove 4 first, the result would be a 5 node vertex cover like(1,4,7,8,9), but if we remove 2 first, the result would be a 4 node vertex cover (2,3,5,6) so this counter case proved that greedy is not always optimal for vertex cover problem

bonus

```

1  function getMultiplyResult(arr[1:n])
2  begin
3      //by defination
4      if (n == 1) then
5          return arr;
6      endif
7      int mid;
8      mid = n/2;
9      //start compute
10     int rightUpArr[1:mid];
11     int leftDownArr[1:mid];
12     //assume we are using deep copy here, start recursion
13     leftDownArr = getMultiplyResult(arr[1:mid]);
14     rightUpArr = getMultiplyResult(arr[mid+1:n]);
15     //leftUpArr,rightDownArr are identity matrix, so should return same arr
    value directly
16     int i;
17     for i=1 to mid do
18         arr[i] += rightUpArr[i];
19         arr[i+mid] += leftDownArr[i];
20     endfor
21     return (arr);
22 end getMultiplyResult
23
24 func main()
25 begin

```



```

26     int x[1:n]=[1,2,3,4,5,6,7,8...]
27     //we can assume n=len(x) here, so n is not passed into function
28     print(getMultiplyResult(x))
29 end main

```

For Time complexity, the algorithm calls it self on half recursively. And will have cn operations per recursive call. So

$$\begin{aligned}
 T(1) &= c \\
 T(n) &= T\left(\frac{n}{2}\right) + cn \\
 \text{Therefore} \\
 T(n) &= O(n \log n)
 \end{aligned}
 \tag{3}$$