1.

a) ×

b) ×

c) ×

2.

a)

pros: simple, and the average complexity is good.

cons: do not guarantee the complexity is always n log(n), at worst case the time complexity will be n^2 .

b)

pros: simple, and the probability of occurring worst case is much more lower than the first case.

cons: also do not guarantee the complexity is always n log(n), in worst case the time complexity will be n^2 .

c)

pros: can guarantee the time complexity is always n log(n).

cons: the algorithm is complicated, and the constant factor of the running time expression is high. When n is not very big, the efficiency of the algorithm is not so good.

3.

1) Array: $O(n^2)$
   Binary Heap: O(n log(n)log(n))
2)

   best: Θ(n)
   worst: Θ(n)

4.

a)

$T(n) = 2T(n/8) + n^{(1/3)}$

log(8,2)= 1/3

case 2 : $n^{(1/3)}\log(n)$

b)

recursion tree is as below:

Tree Time

n ....n

2n/3 ....2n/3

$(2/3)^2$ n..................................... $(2/3)^2$ n

$(2/3)^3$ n...................................... $(2/3)^3$ n

...

1..................................... 1

Depth of recursion tree is Log((3/2),n)

Time complexity : n+2/3n + (2/3)2n + ...+ 1 = n* $(1-(2/3)^{\log((3/2),n)})$/(1-2/3)=3n(1-1/n)=3n-3

So, time complexity is Θ(n)

5.

1) If the majority number exist, then when we divide the whole array into two subarray with same length, at least one of the subarray hold the number as majority.

So if we find a majority number of the subarray , then we iterate the whole array to check the number is whether majority of whole array or not.

If we do not get any majority from two subarray, then the whole array must not contain majority number.

So we can solve the problem using divide and conquer method.

2)

```
majority(A,i,j){
        majority1 = majority(A,i,(j-i)/2);
        majority2 = majority(A,(j-i)/2+1,j);
        if majority1 = majority2="no" then return "no"
        count  <- 0;
        if majority1 != "no"
                then for k <- i to j
                        if A[k] = majority1 then count++;
                if count > (j-i)/2 then return majority1;
        count <-0;
        if majority2  != "no"
                then for k <- i to j
                        if A[k] = majority1 then count++;
                if count > (j-i)/2 then return majority2;
        return "no majority"
}
```

3)

the non-recursive cost is at most 2n .

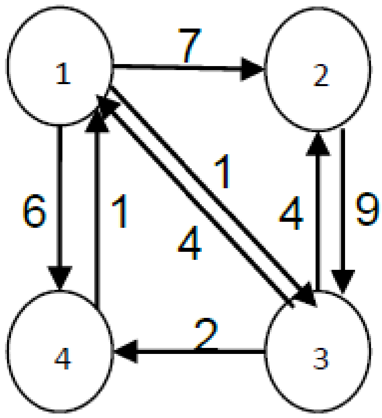
T(n) defined by the recurrence T(1) = 0 and

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

Using master method, can get the time complexity is O(n log n).

6.

D(0)	A	B	C	D
A	0	7	1	6
B	∞	0	9	∞
C	4	4	0	2
D	1	∞	∞	0



D(1)	A	B	C	D
A	0	7	1	6
B	∞	0	9	∞
C	4	4	0	2
D	1	8	2	0

D(2)	A	B	C	D
A	0	7	1	6
B	∞	0	9	∞
C	4	4	0	2
D	1	8	2	0

D(3)	A	B	C	D
A	0	5	1	3
B	13	0	9	11
C	4	4	0	2
D	1	6	2	0

D(4)	A	B	C	D
A	0	5	1	3
B	12	0	9	11
C	3	4	0	2
D	1	6	2	0

7.

Solution: We always choose the customer who has the minimum service time first.

Explanation:

Let execution sequences is $\{a_{k1}, a_{k2}, \ldots, a_{kn}\}$, so the waiting time for

$c_{ki} = t_{k1}+t_{k2}+t_{k3}+\ldots+t_{ki}$.

The total waiting time is then

$1/n\ (c_{k1} + c_{k2} + \ldots + c_{kn})$

$= 1/n\ (t_{k1} + t_{k1} + t_{k2} + t_{k1} + t_{k2} + t_{k3} + \ldots)$

$=1/n\ (n\, t_{k1} + (n-1)t_{k2} + (n-2)t_{k3} + \ldots)$

From the above expression we can easily seen that when the $t_{k1} < t_{k2} < \ldots < t_{kn}$, then the total waiting time is minimum, so we proved our solution can generate correct answer.

* t_{ki} : service time of a_{ki} , c_{ki} waiting time of a_{ki}

8.

1)

$$U(A,1,K) \begin{cases} \text{true} & (A[n] = K); \\ \text{false} & (A[n] \text{ != } K) \end{cases}$$

$$U(A,n,K) \begin{cases} \text{true} & (A[n] = K); \\ U(A,n-1,K-A[n]) \text{ || } U(A,n-1,K) & (A[n] < K) \\ U(A,n-1,K) & (A[n] > K) \end{cases}$$

2)

```
//bottom-up algorithm for getting sub set
Array A <- {a1,a2,…,an};
//table
T[1..n,1..K]
f(A,n,K)
    //init the first row of table
    for i <- 1 to k
        if A[1] = i
            then T[1,i] <- true
            else T[1,i] <- false
        end if
    if T[1,k] = true then return true;
    //init the other rows of table
    for i <- 2 to n
        for j <- 1 to K
            if A[i]=j
                then T[i,j] <- true;
                else if A[i] < j
                    then T[i,j] <- T[i-1,j-A[i]] || T[i-1,j];
                else if A[i] > j
```

```
                                        then T[i,j] <- T[i-1,j];
                    end if
          if T[i,K] = true then return true
//return the final result
return T[n,K]
```