1 (a) The answer is **D**.

Explanation:

A is false. Insertion Sort perform better when the unordered list is almost sorted. Because at every loop will almost did one key comparison which is $O(N)$.

B is partial true but it is not a weak point of Insertion Sort algorithm. In fact, this situation is applicable regardless of any algorithm uses.

C is false. In a new insertion, movements for inserted entries in ordered list are **NOT** always performed. When the entry to be inserted is already bigger than every inserted entries, the movement is not performed.

D is true. This is a weak point of Insertion Sort. Due to when an entry is inserted, it may not in the final position. Therefore, it is possible that it will move to another end of the array.

E is false. The movement in the ordered list (meaning already sorted entries) is done if the current element is less than the previous element.

F is false. When the algorithm completed, the entries definitely is in the correct position.

(b) The answer is **A**

Regardless of it is worst case or best case, merge sort algorithm will always divide the array into half and do comparison on it.

(c) The answer is **C**

C is wrong because Quick Sort Algorithm will partition the list into two sublists based on the pivot element which **may/may not** be the median.

(d) The answer is **F**

The Best case or the Worst case are $O(N \ log \ N)$. Regardless of best case of worst case, deleteMax will always from in $O(log \ N)$

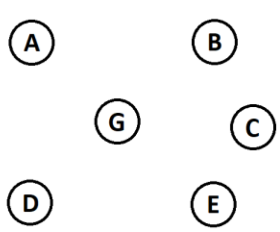
(e) The answer is **E**

Based on the Dijkstra's algorithm, it will be used to find shortest paths from a single source vertex to **ALL** other vertices.

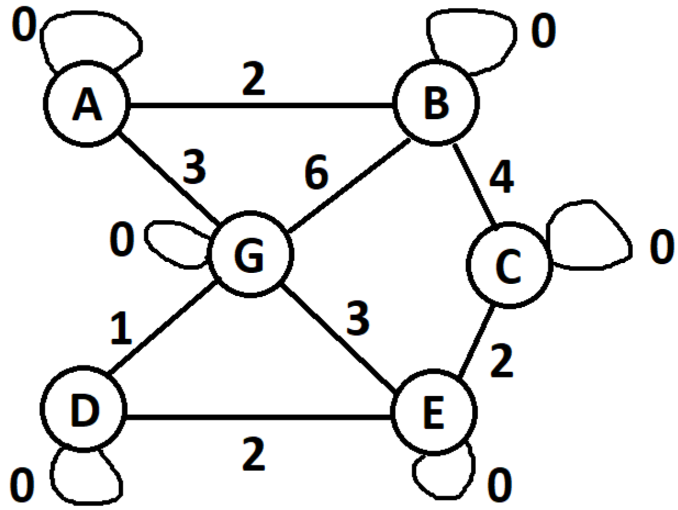
(f) The answer is **B**

B is wrong because it starts off with partitioning the vertices into $|V|$ equilvalence classes (each vertices formed its own equilvalence class), $|V|$ is the number of vertices.

2 _Editor's note:_ The following partial graph is given in the answer book for every part of question 2. The candidates are only required to draw either solid line or dashed line in the answer book.

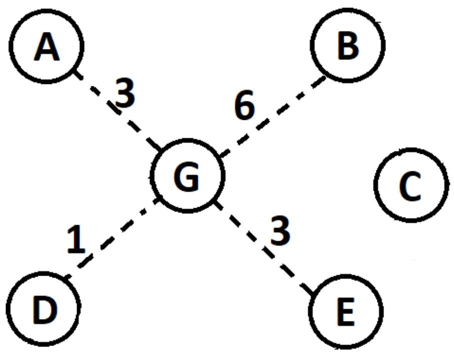


(a)

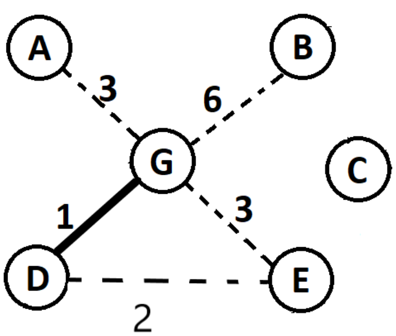


Editor's note: Since there is no mention of it is a simple graph so self loop is possible.

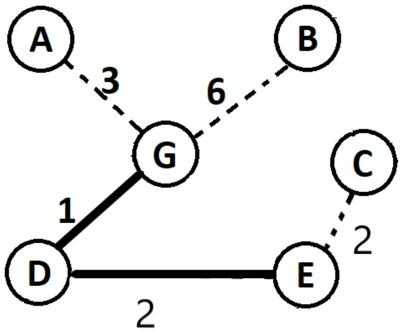
(b)



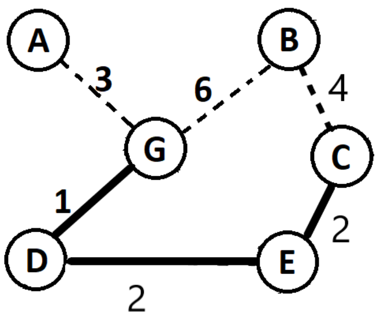
(c)



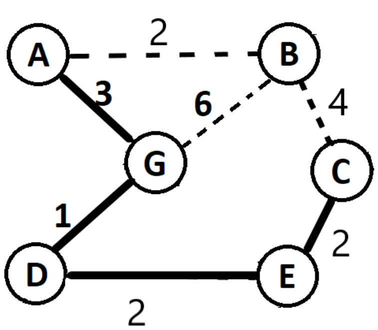
(d)



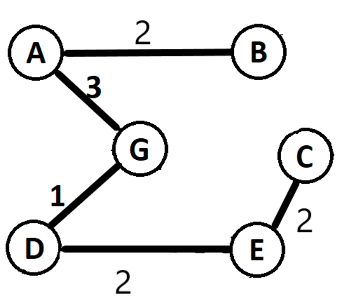
(e)



(f)



(g)



3 (a)

$$W(n) = 3W(n/3) + n - 1$$

By Comparing with the master method formula, $W(n) = aW(n/b) + f(n)$

$$a = 3$$
$$b = 3$$
$$f(n) = n - 1$$
$$n^{\log_b a} = n^{\log_3 3} = n^1$$
$$f(n) = \theta(n) = \theta\left(n^{\log_b a}\right)$$
$$\text{Since f(n)} = \theta\left(n^{\log_b a}\right),$$
$$\text{By using second formula of Master Method,}$$
$$\text{W(n)} = \theta\left(n^{\log_b a}\, log\, n\right) = \theta(n\, log\, n)$$

(b)

$$a_n = 9a_{n-2}$$
$$a_n = 0\, a_{n-1} + 9a_{n-2}$$

The characteristic equation of the recurrence equation is

$$x^2 = 0\, x + 9$$
$$x^2 = 9$$
$$x = \pm 3$$

The characteristic root is ± 3.

Therefore, the Recurrence equation is

$$a_n = a(3)^n + b(-3)^n\, , a\ and\ b\ are\ constants$$

By substituiting the initial condition

$$a_0 = 1$$
$$a(3)^0 + b(-3)^0 = 1$$
$$a + b = 1 \quad \ldots.(eq\ 1)$$

$$a_1 = 3$$
$$a(3)^1 + b(-3)^1 = 3$$
$$3a - 3b = 3$$
$$a - b = 1 \quad \ldots.(eq2)$$

eq 1 + eq 2

$$2a = 2$$
$$a = 1$$

Sub a = 1 into eq 1

$$1 + b = 1$$
$$b = 0$$

Therefore, finally

$$a_n = 1\,(3)^n + 0\,(-3)^n$$
$$a_n = 3^n$$

(c) (i) charJump['A'] = 0
charJump['B'] = 4
charJump[Other Char] = 5

matchJump

Pattern	B	A	A	A	A
matchJump	9	4	4	4	1

(ii) Assume that the string is 1-indexed
Text (T) : AAAAAAAAAAAAAAAAAAAA
Pattern (P) : BAAAA

Let assume a simple case where text consists of 5 'A's
Pattern: BAAAA
Text : AAAAA
The Boyer-Moore algorithm will compare from the last index of the Pattern until the first mismatch.
5 comparison will be made (including mismatch index)

Back to the question where there are 20 'A's. We use the Boyer-Moore algorithm, we align the pattern with the text first as shown below.
Pattern: BAAAA
Text : AAAAA…AAAAA (20 'A's)

When there is a mismatch, we will shift the pattern toward right.

Since every time when we align the pattern with any index of the text (between 5 and 20 inclusive), It will be same as the simple case I mentioned above. Therefore, the main point will be how many times we will shift the pattern (let say x)? The total number of key comparison will be $5 \times x$.

<u>Simple Boyer-Moore algorithm</u>
Recall that from the lecture notes
J is the index of the element on the text
K is the index of the element on the pattern.
M is the length of the pattern = 5
When there is a mismatch, the following shift will occur.

```
      j+=max(charJump[T[j]],m-k+1);
charJump[T[j]] = charJump['A'] = 2
m-k+1 = 5 – 1 + 1 = 5.
max(charJump[T[j]],m-k+1) = max(2,5) = 5
```

Therefore j will add 5

P (After Shift)		B	A	A	A	A
P	B	A	A	A	A	
T	A	A	A	A	A	A
Index	1	2	3	4	5	6
	Mismatch happen, j=1			After shift, j = 6		

Therefore the pattern is shift by 1 space towards right.
The following diagram show how the shifting works

```
             BAAAA
            BAAAA
           ….
   BAAAA
  BAAAA
BAAAA
AAAAAAAA…AAAAA
            (Stop the shift at j=16)
```

In total, the pattern will be occur (16-1+1) = 16 times.
Answer: 16 * 5 = **80 comparisons**

Boyer-Moore algorithm

Recall that from the lecture notes
J is the index of the element on the text
K is the index of the element on the pattern.
M is the length of the pattern = 5
When there is a mismatch, the following shift will occur.

```
       j+=max(charJump[T[j]],matchJump[k]);
charJump[T[j]] = charJump['A'] = 2
matchJump[k] = matchJump[1] = 9.
max(charJump[T[j]],m-k+1) = max(2,9) = 9
```

Therefore j will add 9

P (Shift)						B	A	A	A	A		
P	B	A	A	A	A							
T	A	A	A	A	A	A	A	A	A	A	A	A
Index	1	2	3	4	5	6	7	8	9	10	11	12
	Mismatch happen, j=1						After Shift, j = 10					

Therefore the pattern is shift by 5 space towards right.
The following diagram show how the shifting works.

```
                    BAAAA
               BAAAA
          BAAAA
     BAAAA
AAAAAAAAAAAAAAAAAAAA
```

In total, the pattern will occur 4 times

Answer: 4 * 5 = **20 comparisons**

4 (a) (i) Assume that the string is 1-indexed

$$\text{Alignment(n1, n2)} \begin{cases} n2\,,n1\ = 0 \\ n1\,,n2\ = 0 \\ alignment\ (n1\ - 1, n2\ - 1)\,, S[n1] == S[n2] \\ min(\ alignment\ (n1\ - 1, n2)\,, alignment\ (n1, n2 - 1)\ ) + 1\,,\ otherwise \end{cases}$$

(ii)

$$\begin{tikzcd}
alignment(3,4) \arrow[r] \arrow[d] \arrow[dr] & alignment(2,4) \arrow[r] \arrow[d] \arrow[dr] & alignment(1,4) \arrow[r] \arrow[d] \arrow[dr] & alignment(0,4) \arrow[d] \\
alignment(3,3) \arrow[r] \arrow[d] \arrow[dr] & alignment(2,3) \arrow[r] \arrow[d] \arrow[dr] & alignment(1,3) \arrow[r] \arrow[d] \arrow[dr] & alignment(0,3) \arrow[d] \\
alignment(3,2) \arrow[r] \arrow[d] \arrow[dr] & alignment(2,2) \arrow[r] \arrow[d] \arrow[dr] & alignment(1,2) \arrow[r] \arrow[d] \arrow[dr] & alignment(0,2) \arrow[d] \\
alignment(3,1) \arrow[r] \arrow[d] \arrow[dr] & alignment(2,1) \arrow[r] \arrow[d] \arrow[dr] & alignment(1,1) \arrow[r] \arrow[d] \arrow[dr] & alignment(0,1) \arrow[d] \\
alignment(3,0) \arrow[r] & alignment(2,0) \arrow[r] & alignment(1,0) \arrow[r] & alignment(0,0)
\end{tikzcd}$$

__Editor's Note__: Here I does not assume the case of the example given in the question, I was plotting the subproblem graph in general.

(iii) The required code is as below:

```
int alignment (int n1, int n2):
      S1 = "0" + S1;
      S2 = "0" + S2;
      //Since S1 & S2 are 1-indexed so I convert them into 1-indexed
      int dp[n1+1][n2+1];
      for (int i=0; i<= n1 ; i++)
            for (int j=0; j<=n2; j++)
                  if (i==0)               dp[i][j] = j;
                  else if (j==0)          dp[i][j] = i;
                  else if (S1[i]==S2[j])  dp[i][j] = dp[i-1][j-1];
                  else
                  dp[i][j] = min(alignment(i-1,j),alignment(i,j-1)+1
      return dp[n1][n2]
```

(b) We calculate the P/S value for all the object

P (profit)	10	40	35	45	6
S (size)	5	4	7	5	3
P/S	2	10	5	9	2

and sort them by P/S value in decreasing order (if there is a ties, we sort by profit desc. order).

P (profit)	40	45	35	10	6
S (size)	4	5	7	5	3
P/S	10	9	5	2	2

Next, we greedily choose the object in the decreasing order of P/S provided the total of chosen objects doesn,t exceed the maximum limit (20). In other word, we will choose the object with highest P/S value if the total size + size of the object <= 20. If not we will chose the next object.

Iteration	Current Profit	Current Size	P/S of Chosen object	Profit of Chosen Object	Size of Chosen Object	Can I choose it?	After Profit	After size
1	0	0	10	40	4	YES	40	4
2	40	4	9	45	5	YES	85	9
3	85	9	5	35	7	YES	120	16
4	120	16	2	10	5	NO	120	16
5	120	16	2	6	3	YES	126	19

The maximum profit is **126**

Time complexity analysis:

Calculating the P/S value linear scan though the array O(n)
Sorting the array according to P/S value (assume mergeSort algorithm) O(n log n)
Iterate through each object required O(N), decision on choosing the object O(1) :: O(N)

In Total The time complexity is O(N + N log N + N) = O(N log N)

Editor's note: If you assume others sorting algorithm like Insertion Sort. The answer will be $O(N^2)$

Solver: Lee Zong Yu (ZLEE043@e.ntu.edu.sg)