Question 4. [20 marks]] Let X and Y be sets of n numbers X={x_1, …, x_n} and Y={y_1, …, y_n}. The set of n2 numbers {$x_i + y_j$ | i=1, …, n; j=1, …, n} is called the Cartesian sum of X and Y and denoted by X + Y. We want to compute the smallest n elements of X + Y.

The easiest way for this is to sort X and Y first, and expand X + Y into two dimensional array. Then merge the first row with the second, take the first n elements. Then repeatedly merge rows and take the first n elements. This process takes $O(n^2)$ time, as the time for merge is O(n) and we merge n-1 times. Obviously this can be improved.

We merge X and Y first. Then obviously $x_1 + y_1$ is the first element of the solution. The candidate for the second is x_2+y_1 or x1+y2. In general, we call xi+1 + xj the lower neighbor of $x_i + y_j$, and x_i+y_j+1 the righ neghbour. We also call them adjacent elements. The selection process is similar to that of Dijkstra's algorithm for the single source shortest path problem with a heap data structure. As we select a new sum from a heap, we can say we move it from the frontier to the solution set. Then we can insert adjacent elements into the heap. An example and an abstract algorithm is shown below.

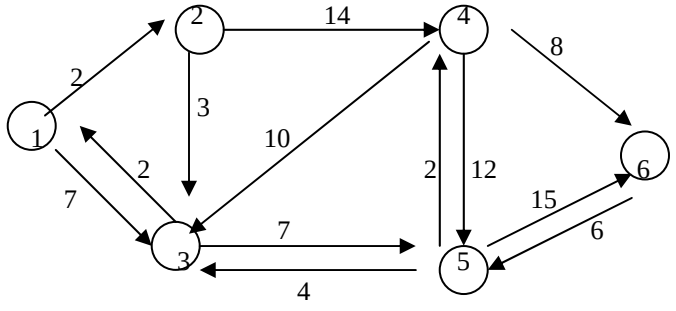
```
           1  2  3  4  5
X  / Y     1  6 11 12 15
-----------------------
1  1       2  7 12             Let (2, 5, 7, 9) be selected
2  4       5 10                frontier = {10, 11, 12, 15}
3  8       9 15
4 10      11
5 19

sort X and Y
for kk=1 to n do begin
  a[kk]=delete_min(HEAP)    // a is the container for solution
  i=a[kk].i; j=a[kk].j
  if(i+1, j) has not been in HEAP
    insert xi+1 + yj with indices (i+1, j) into HEAP
  if (i, j+1) has not been in HEAP
    insert xi + yj+1 with indices (I, j+1) into HEAP
end
```

(a) The following is a partially completed C program. Fill the blank parts referring to the labels. The function "siftup" is to make the data structure back into a heap when the root violates the heap condition. Parameter min=0 is for the minimum heap and max=1 is for the maximum heap. The minimum heap maintains the minimum at the root and the maximum heap is for the maximum. In the function, the key comparison is controlled by the flag with the exclusive-or operator ^. The heap is maintained in the array Z.

```
struct item{
  int key; int i; int j;
};
int i,n,t,k, size, min=0, max=1, F[100][100];
struct item X[100], Y[100], Z[100], a[100];
siftup(int p, int q, struct item a[], int flag)
{ int j,k; struct item y, z;
  y=a[p]; j=p; k=2*j;
  while (k<=q){z = a[k];
    if (k<q) if (flag^(z.key>a[k+1].key)){k++; z = a[k];}
    if (flag^(y.key<=z.key)) break;  // ^ is for exclusive or
    a[j] = z; j = k; k = 2*j;
  } a[j]=y;
}
sort(int n, struct item a[])
```

```
{ int i; struct item w;
  for (i=n/2; i>=1; i--) siftup(i,n,a,max);
  for (i=n-1; i>=1; i--) {
    w = a[i+1]; a[i+1] = a[1]; a[1] = w;
    siftup(1,i,a,max);
  }
}
insert(struct item x, struct item a[]){
  int i,j; struct item y;
  size++; i=size;
  F[x.i][x.j]=1;
  while(i>=2){ j=i/2; y=a[j];
    if(x.key>=y.key) break;
    a[i]=y; i=j;
  } a[i]=x;
}
struct item delete_min(struct item b[]){
  struct item w= b[1];
  b[1]=b[size]; size--; siftup(1, size, b, min);
  return w;
}
int f(int i, int j){ return F[i][j]; }
select(int n, struct item X[], struct item Y[], struct item Z[]){
  int i, j, ii, kk, size=1;
  struct item w;
  for(kk=1; kk<=k; kk++){
    a[kk]=delete_min(Z);
    i=a[kk].i; j=a[kk].j;
    if(f(i+1,j)==0){w.key=X[i+1].key + Y[j].key; w.i=i+1; w.j=j;
    insert(w, Z); }
    if(f(i,j+1)==0){w.key=X[i].key + Y[j+1].key; w.i=i; w.j=j+1;
    insert(w, Z); }
  }
}
main()
{  struct item w;
  printf("input size \n "); scanf("%d",&n); getchar();
  size=0; k=n;
  for (i=1;i<=n;i++)X[i].key=random()%21;
  for (i=1;i<=n;i++)Y[i].key=random()%20;
  sort(n, X); sort(n, Y);
  w.key=X[1].key+Y[1].key; w.i=1; w.j=1; insert(w, Z);
  for (i=1;i<=n;i++) X[i].i=i; for (i=1;i<=n;i++) Y[i].i=i;
  select(n, X, Y, Z);
}
```

Question 5. [20 marks for the whole question] A directed graph is given below.

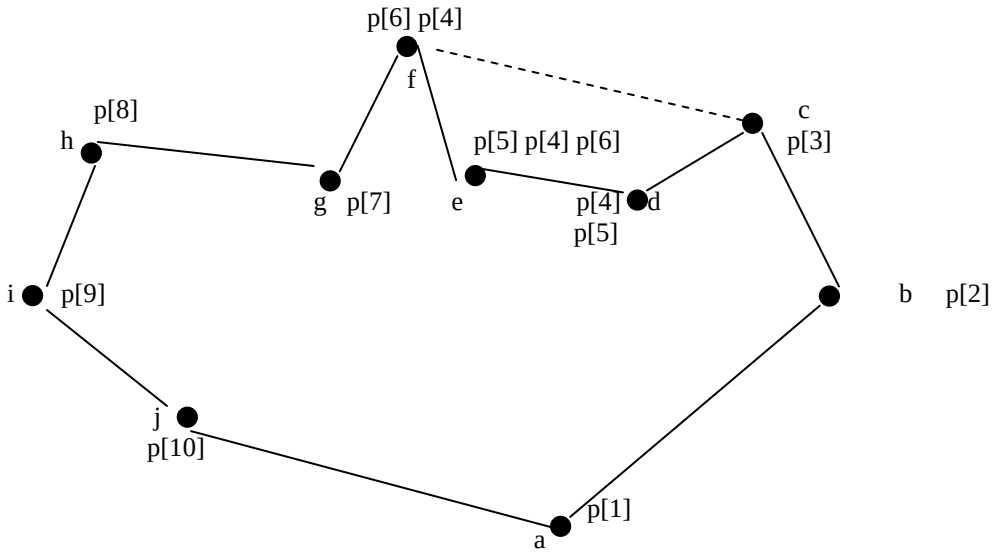


(a) Trace Dijkstra's algorithm for the single source shortest paths with vertex 1 as the source. [10 marks]

(b) Trace Floyd's algorithm for all pairs shortest paths [10 marks]

Question 4. The following is a trace of Graham's algorithm for a convex hull. The working is partially done. Complete the trace. [20 marks]

In the following picture, the history of p[i] is shown horizontally to the right or vertically down as time proceeds. For example, e is originally in p[5], but contained in p[4] at the next step, etc.



Array p is the container of points, initialized to (p[1], …, p[10])=(a, b, c, d, e, f, g, h, h, i, j).

The following is the main part of Graham's algorithm. The value of m returned at the end gives the number of vertex points of the convex polygon. Angles are measured from the x-axis. The following algorithm starts after the angles from p[1] to all other points are sorted, that is, angles of (p[1], p[2]), (p[1], p[3]), … are in non-decreasing order. For simplicity, we assume there are no three points on a straight line. The value of turn(p[m], p[m-1], p[i]) is -1 if three points (p[m-1], p[m], p[i]) in this order make a left turn, and -1, otherwise.

```
m = 3;
for (i=4; i<=n; i++) {
   while (turn(p[m], p[m-1], p[i]) >= 0) m--;
   m++;
   t = p[m];  p[m] = p[i]; p[i] = t;
}
return m;
```

Each point is given by the x and y co-ordinates.
The snapshots are at the end of each iteration with i, that is, immediately after the swapping.

We test left turn or right turn immediately inside the for-loop, using current i and previous m, that is, the value of i at the i-th row and that of m at the (i-1)-th row.

Example. At the 5-th row, take (p[5], p[4], p[3]), right turn seen from p[3]. Decrease m and increase m, then swap p[5] and p[4].
At the 6-th row, take (p[6], p[4], p[3]), right turn seen from p[3]. Decrease m and Increase m, then swap p[6] and p[4]

i	m	p	1	2	3	4	5	6	7	8	9	10	
	3		a	b	c	d	e	f	g	h	i	j	before for-loop with i.
4	4		a	b	c	d	e	f	g	h	i	j	
5	4		a	b	c	e	d	f	g	h	i	j	
6	4		a	b	c	f	d	e	g	h	i	j	

Choose Question 6 or 7 for the last question . If you answer both, the better one will be marked.

Question 6. [15 marks] The following is the KMP algorithm for pattern matching and its trace with an example. Some parts of the C code are omitted.

```
int mismatch(char a, char b){
  c++;
  if(a!=b) return 1; else return 0;
}
main(){
t=0; h[1]=0;
for(i=2; i<=m; i++){
  while((t>0)&&(pat[i-1]!=pat[t])) t=h[t];
  t++;
  printf("i, t  %d %d  pat[i] pat[t] %c %c  ",i, t, pat[i], pat[t]);
  if (pat[i]!=pat[t])h[i]=t; else h[i]=h[t];
  for(j=1;j<=i;j++)printf("%d ",h[j]); printf("\n");
}
for(i=1;i<=m; i++) printf("%d ",h[i]);
printf("\n");
for(j=1; j<=n; j++) printf("%c ", text[j]);
printf("\n");
j=1; i=1; c=0;
out(1);
while(i<=m && j<=n){
  while((i>0)&&mismatch(pat[i],text[j])){i=h[i]; out(i);}
  i++; j++;
}
  if (i>m)printf("found at %d\n", j-i+1);
  else printf("not found\n");
  printf("count= %d\n", c);
}
out(int k){ int i;
  for(i=1; i<=j-k; i++)printf("  ");
  for(i=1; i<=m; i++)printf("%c ",pat[i]);
  printf("\n");
}
/* sample session
```

```
input m 7
input pat abbabbc
input n 17
input text abbabbabbaabbabbc
a b b a b b c
i, t  2 1  pat[i] pat[t] b a  0 1
i, t  3 1  pat[i] pat[t] b a  0 1 1
i, t  4 1  pat[i] pat[t] a a  0 1 1 0
i, t  5 2  pat[i] pat[t] b b  0 1 1 0 1
i, t  6 3  pat[i] pat[t] b b  0 1 1 0 1 1
i, t  7 4  pat[i] pat[t] c a  0 1 1 0 1 1 4
0 1 1 0 1 1 4
a b b a b b a b b a a b b a b b c
a b b a b b c
      a b b a b b c
            a b b a b b c
                  a b b a b b c
found at 11
count= 20
*/
```

Following this example, trace the algorithm with pattern = abcabac and text = abcabcababababcabac

Question 7. [15 marks] Trace the Euclidean algorithm for the greatest common divisor and multiplicative inverse for a=119 and b=77. Show the greatest common divisor, A^{-1} mod B and B^{-1} mod A, where A and B are given by a=gcd(a,b)*A and b=gcd(a,b)*B..