Q1: we suppose that we will sort A[1…n] In merge sort. Because we have points in plane then each point have his coordinates p1= (xi,yi) when we compare it to find the identical point we need to compare it coordinates with the other points. Suppose we have p2(xi,yi). To decide if the p1 is identical point to p2 then first we need to compare only x coordinate first if p1(xi)=p2(xi) then we will compare p1(yj) with p2 (yj). If p1(yj)=p2(yj) then p1 is identical point to p2. The run time for merge sort is O(n lg n).

Q2: first, we need to choose every point p in turn,to handl them as origin. Second, we will need to sort other points depend on polar angle for the points. We will use merge sort, and we will compare the polar angles of q, r wrt p by considering the vectors −pq→ and −pr→. If (r − p) × (q − p) < 0, then r has greater polar angle wrt p than q. if (r − p) × (q − p) > 0, then q has polar angle greater than r if r − p has a non-negative y-coordinate. . finally, we will use linear search of the sorted points. If two points have equal polar angles, then they are co-linear, and we return 'yes'.

Q3:
First , we will need to sort A,then we will assign U= A. after that we will do for loop

```
1.For I to n do
  11.use binary search to chck if B[i] belong to A or not
 1.2:if B[i] does not belong to A, add B[i] to U,
 2. return(u)
```

Q4:

6	4	9	8	5	10	1	3

Pivot =3

I=0 , j=1

1	3	9	8	5	10	6	4

1

3

9	8	5	10	6	4

Pivot=4 , i=0,j=1

4	8	5	10	6	9

4

8	5	10	6	9

Pivot=9 , i=0,j=1

8	5	6	9	10

8	5	6

9

10

8	5	6

Pivot=6 , i=0,j=1

5	6	8

5

4

8

-the final result will be

1	3	4	5	6	8	9	10

Q5: first we will use binary search; we will repeat these steps with each comparison.
First, we will pick the middle index (first +last%2). We will check if mid is equal to the element stored at mid. If it is true we will return mid. If the mid less than the element we are looking for then we will ignore the elements that less than mid , if the mid greater than the element we are looking for then we will ignore all element that greater than mid. The run time of the algorthim will be the run time of biary search Θ(lg n) time.

```
Search(A, i, j)
if i <= j do
mid <-- (i+j)/2;
if A[mid] = mid then
return(mid);
if A[mid] > mid then
return (Search(A, i, mid-1));
if A[mid] < mid then
return(Search(A, mid+1, j));
else return(Nil);
```

Q6:
A:we can sorted array has N elements in running time $\Theta(k^2)$ by using Insertion-Sort. If we have n/k array it can be sorted in time $(n/k)\Theta(k^2) = \Theta(nk)$.

B: there are many levels of merging. In level 0 we will have n/k subarray. We group the subarray into groups each containing two subarrays, and we merge them by using merge sort algorithm. the run time for this algorithm in level 0 is O(n).in level 1 we will have n/2k subarray, and also We group the subarray into groups each containing two subarrays, and we merge them by using merge sort algorithm. we will apply the same algorithm until we have one sorted array. The number of subarray are reducing in each level by half, then the run time is lg (n/k). The total running time will be O(n lg (n/k)).

C: k = Θ(lg n).