**1. Insertion Sort**

1 #include <stdio.h>

2

3

4

5 **void** insertion\_sort(**int array**[],**int size**)

6 {

7 **int** i,j,item;

8 **for**(i=1;i<**size**;i++)

9 {

10 item=**array**[i];

11

12 j=i-1;

13 **while**( j>=0 && **array**[j]>item)

14 {

15

16 **array**[j+1]=**array**[j];

17 j--;

18

19 }

20 **array**[j+1]=item;

21 }

22 }

23

24 **int** main()

25 {

26 **int** i;

27 **int array**[]={9,1,3,5,8,4,7};

28

29 insertion\_sort(**array**,8);

30 **for**(i=0;i<7;i++)

31 printf("%d",array[i]);

32 return 0;

33 }

**2-3.Quick Sort-Merge Sort**

1 #include<bits/stdc++.h>

2 #include<stdlib.h>

3 **using namespace std**;

4

5

6

7

8 **void** quick\_sort(**int array**[],**int** f , **int** l)

9 {

10 **int** i , j,temp;

11 **if**(f < l)

12 {

13 i = f+1;

14 j = l;

15 **while** (**array**[i] < **array**[f])

16 {

17 i++;

18 }

19 **while**(**array**[j] > **array**[f])

20 {

21 j--;

22 }

23 **while**(i < j)

24 {

25

26 temp=**array**[j];

27 **array**[j]=**array**[i];

28 **array**[i]=temp;

29

30 **while** (**array**[i] < **array**[f])

31 {

32 i++;

33 }

34

35 while(array[j] > array[f])

36 {

37 j--;

38 }

39 }

40

41 temp=array[f];

42 array[f]=array[j];

43 array[j]=temp;

44

45 quick\_sort(array,f , j-1);

46 quick\_sort(array,j+1 , l);

47 }

48 }

49

50

51 void arrayMerge(int array[] , int left , int mid , int right)

52 {

53 int n1 = mid-left+1;

54 int n2 = right-mid;

55 int L[n1+1] , R[n2+1];

56 for(int i = 0 ; i < n1 ; i++)

57 {

58 L[i] = array[left+i];

59 }

60 for(int j = 0 ; j < n2 ; j++)

61 {

62 R[j] = array[mid+1+j];

63 }

64 L[n1] = INT\_MAX;

65 R[n2] = INT\_MAX;

66

67 for(int i = 0 , j = 0 , k = left ; k <= right ; k++)

68 {

69 if(L[i] <= R[j])

70 {

71 array[k] = L[i++];

72 }

73 else

74 {

75 array[k] = R[j++];

76 }

77 }

78 }

79

80 void merge\_sort(int array[] , int left , int right)

81 {

82 if(left<right)

83 {

84 int mid = floor((left+right)/2);

85 merge\_sort(array , left , mid);

86 merge\_sort(array, mid+1 , right);

87 arrayMerge(array, left , mid , right);

88 }

89 }

90

91 int main()

92 {

93 int n;

94 printf("Enter Size Of Array: ");

95 scanf("%d",&n);

96 int array[60];

97 for(int i = 0 ; i < n ; i++)

98 scanf("%d",&array[i]);

99

100

101 //quick\_sort(array,0 , n-1);

102 merge\_sort(array , 0 , n-1);

103

104 printf("After Sorting: ");

105 for(int i = 0 ; i < n ; i++)

106 printf("%d ",array[i]);

107

108

109 }

**4.Heap Sort**

1 #include <stdio.h>

2

3 **int** left(**int** i)

4 {

5 **return** 2\*i;

6 }

7 **int** right(**int** i)

8 {

9 **return** 2\*i+1;

10 }

11 **int** parent(**int** i)

12 {

13 **return** i/2;

14 }

15

16 **void** max\_heapify(**int** heap[],**int** heap\_size,**int** i)

17 {

18 **int** l,r,largest,t;

19 l=left(i);

20 r=right(i);

21

22 **if**(l<=heap\_size && heap[l]>heap[i])

23 largest=l;

24 **else**

25 largest=i;

26 **if**(r<=heap\_size && heap[r]>heap[largest])

27 largest=r;

28

29 **if**(largest != i)

30 {

31 t=heap[i];

32 heap[i]=heap[largest];

33 heap[largest]=t;

34 max\_heapify(heap,heap\_size,largest);

35 }

36 }

37 **void** build\_maxheap(**int** heap[],**int** heap\_size)

38 {

39 **int** i;

40 **for**( i=heap\_size/2 ; i>=1 ; i--)

41 {

42 max\_heapify(heap,heap\_size,i);

43 }

44 }

45 **void** heap\_sort(**int** heap[],**int** heap\_size)

46 {

47 **int** i,t;

48 **for**(i=heap\_size;i>1;i--)

49 {

50 t=heap[1] ;

51 heap[1]=heap[i];

52 heap[i]=t;

53

54 heap\_size -= 1;

55 max\_heapify(heap,heap\_size,1);

56 }

57 }

58

59 **int** main (**void**)

60 {

61 **int** heap[6]={0,4,3,1,5,8};

62 build\_maxheap(heap,5);

63 heap\_sort(heap,5);

64 **int** i;

65 **for**(i=1;i<=5;i++)

66 printf("%d",heap[i]);

67 }

**5.Priority Queue**

1 #include <stdio.h>

2

3 **int** left(**int** i)

4 {

5 **return** 2\*i;

6 }

7 **int** right(**int** i)

8 {

9 **return** 2\*i+1;

10 }

11 **int** parent(**int** i)

12 {

13 **return** i/2;

14 }

15

16 **void** max\_heapify(**int** heap[],**int** heap\_size,**int** i)

17 {

18 **int** l,r,largest,t;

19 l=left(i);

20 r=right(i);

21

22 **if**(l<=heap\_size && heap[l]>heap[i])

23 largest=l;

24 **else**

25 largest=i;

26 **if**(r<=heap\_size && heap[r]>heap[largest])

27 largest=r;

28

29 **if**(largest != i)

30 {

31 t=heap[i];

32 heap[i]=heap[largest];

33 heap[largest]=t;

34 max\_heapify(heap,heap\_size,largest);

35 }

36 }

37 void build\_maxheap(int heap[],int heap\_size)

38 {

39 int i;

40 for( i=heap\_size/2 ; i>=1 ; i--)

41 {

42 max\_heapify(heap,heap\_size,i);

43 }

44 }

45 int extract\_max(int heap[],int heap\_size)

46 {

47 int i,maxitem;

48 maxitem=heap[1];

49

50 heap[1]=heap[heap\_size];

51 heap\_size--;

52 max\_heapify(heap,heap\_size,1);

53

54 return maxitem;

55 }

56 int main (void)

57 {

58 int heap[6]={0,4,3,1,8,5};

59 build\_maxheap(heap,5);

60 printf("%d",extract\_max(heap,5));

61 //printf("%d",heap[i]);

62 }