Date
Page

① HEAP SORT

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void bottom_heapify (int n, int a[]);
void heapsort (int n, int a[]);
void printArray (int a[], int n);

int main()
{
    int a[15000], n, i, j, ch, temp;
    clock_t start, end;

    while(1) {
        printf("\n1: Manual Entry");
        printf("\n2: To display time taken for sorting
                  number of elements N from 500 to
                  14500");
        printf("\n3: To exit");
        scanf("%d", &ch);
        ~~printf("Enter~~
        switch (ch)
        {
            case 1: printf("Enter no. of elements:");
                    scanf("%d", &n);
                    printf("Enter array elements:");
                    for (i=0; i<n; i++)
                    {
                        scanf("%d", &a[i]);
                    }
                    start = clock();
```

[illegible] 20/6/24

```
heapsort (n, a);
end = clock();
printf ("\n sorted array :");
printf( a, n);
printf ("\n Time taken to sort
      %d numbers is %f sec",n,
   (((double) (end-start))/CLOCKS_PER_SEC));
break;

case 2:
   n= 500;
   while (n <=14500)
   {
        for(i= 0;  i<n; i++)
        {
            a[i] =n-i;
        }
        start= clock();
        heapsort (n, a);
        for (j=0;  j <5000000; j++)
        {
            temp = 38/600;
        }
        end = clock();
      printf ("\n Time taken");
      n= n+1000;
   }
   break;
```

```
        case 3 :  exit (0);
                  break;

       default: printf ("\n Invalid choice! Try again.");
    }
    getchar();
   }
   return 0;
}


void  bottom_heapify  (int n, int a[])
{
      int p, item, c;
      for ( p = (n-1) /2 ;  p>=0 ; p--)
      {
            item= a[p];
            c=2* p+1;
            while ( c<= n-1 && a[c] <a [c+1])
            {
                  c++;
            }
            if (item <a[c])
            {
                  a[p]=a[c];
                   p=c;
            }
            else
                  break;
            c=2*p+1;
      }
      a[p]= item;
  }
}
```

```
void   heapsort (int n, int a[])
  {
      bottom_ heapify (n, a);

    for (int i = n-1;  i > 0;  i--)
    {
         int temp = a[0];
         a[0] = a[i]
         a[i] = temp;
         bottom_ heapify  (i, a);
    }
  }


void  printArray  (int a[], int n)
    for (int i = 0;   i < n;  i++)
    {
          printf( "%d" ,  a[i]);
    }
 }
```

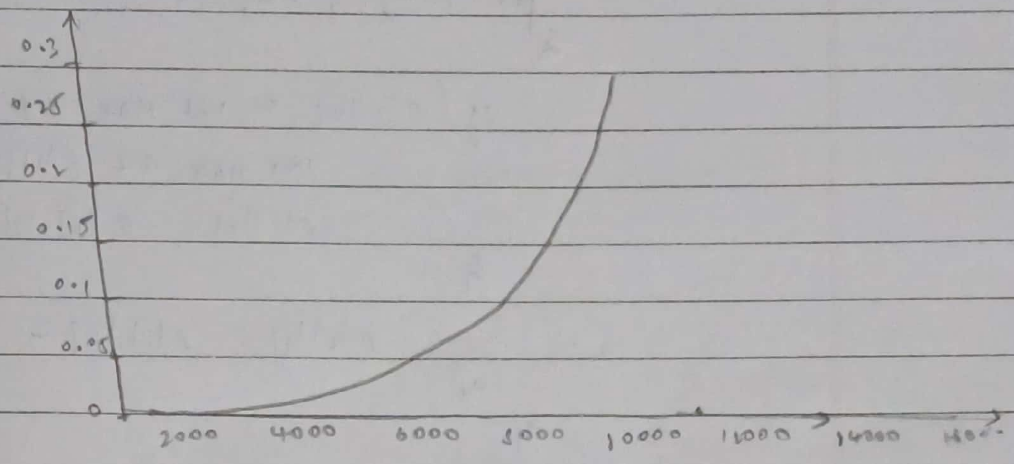
→ output

```
1:  For manual  entry
2:    Random  generator
3:   To  exit

Enter choice : 1
Enter no. of  elements :  7
```

Enter array elements: 4, 3, 1, 6, 2

Sorted Array: 1 2 3 4 6



② → Floyds

```
#include <stdio.h>
#include <limits.h>

void floyd (int n, int cost [] [n], int D [] [n])
{
        int i, j, k;
      for (i=0; i<n; i++)
      {
           for (j= 0; j <n; j++)
           {
                  D [i][j]= cost [i][j];
           }
      }
```

[illegible] 20/6/24

```
for(k=0; k<n ; k++)
{
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            if(D[i][k]!= int_MAX && D[k][j]!=
                    INT_MAX && D[i][j] >
                    D[i][k] + D[k][j])
            {
                D[i][j] = D[i][k] + D[k][j];
            }
        }
    }
}
}

void printShortestPath(int n, int D[][n])
{
    printf("Shortest path \n");
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            if(D[i][j] == INT_MAX)
            {
                printf("INF\t");
            }
            else
            {
                printf("%d\t", D[i][j]);
            }
        }
    }
```

```
int main()
{
    int n;
    printf ( "Enter no. of vertices ");
    scanf ( "%d", &n);

  int cost [n][n]
  printf( "Enter the cost adjacency matrix \n");
  for (int i=0; i<n; i++)
  {
        for (int j=0; j<n; j++)
        {
              scanf ( "%d", &cost [i][j]);
              if (cost [i][j] == -1)
              {
                    cost[i][j]==-1)
                    {
                          cost[i][j]= int_MAX;
                    }

              }
        }

        int D [n][n];
        floyd (n, cost, D);
        printShortestPath (n, D);
        return 0;
}
```