ALGORITHMS LABORATORY
[CS-2098]
Individual Work

Lab. No.- 5 Date. 05/10/2021

Roll Number:	1905083	Branch/Section:	CSE/CS-2
Name in Capital:	AMIL UTKARSH GUPTA		

(Instruction: Rename this file as r-LAB NO-x where r is your roll number & x is your lab. number & Suppose your roll number is 1905123 & you want to submit lab-2 programs, then file name should be 1905123-LAB No-2. Finally delete all texts inside parentheses, also parenthesis)

Program No: 5.1
Program Title:

Write a menu (given as follows) driven program to sort an array of n integers in ascending order by heap sort algorithm and perform the operations on max heap. Determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the array to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

MAX-HEAP & PRIORITY QUEUE MENU
0.
Quit
1.
n Random numbers=>Array
2.
Display the Array
3.
Sort the Array in Ascending Order by using Max-Heap Sort technique
4.
Sort the Array in Descending Order by using any algorithm
5.
Time Complexity to sort ascending of random data
6.
Time Complexity to sort ascending of data already sorted in ascending order
7.
Time Complexity to sort ascending of data already sorted in descending order
8.
Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000
9.
Extract largest element
10.
Replace value at a node with new value

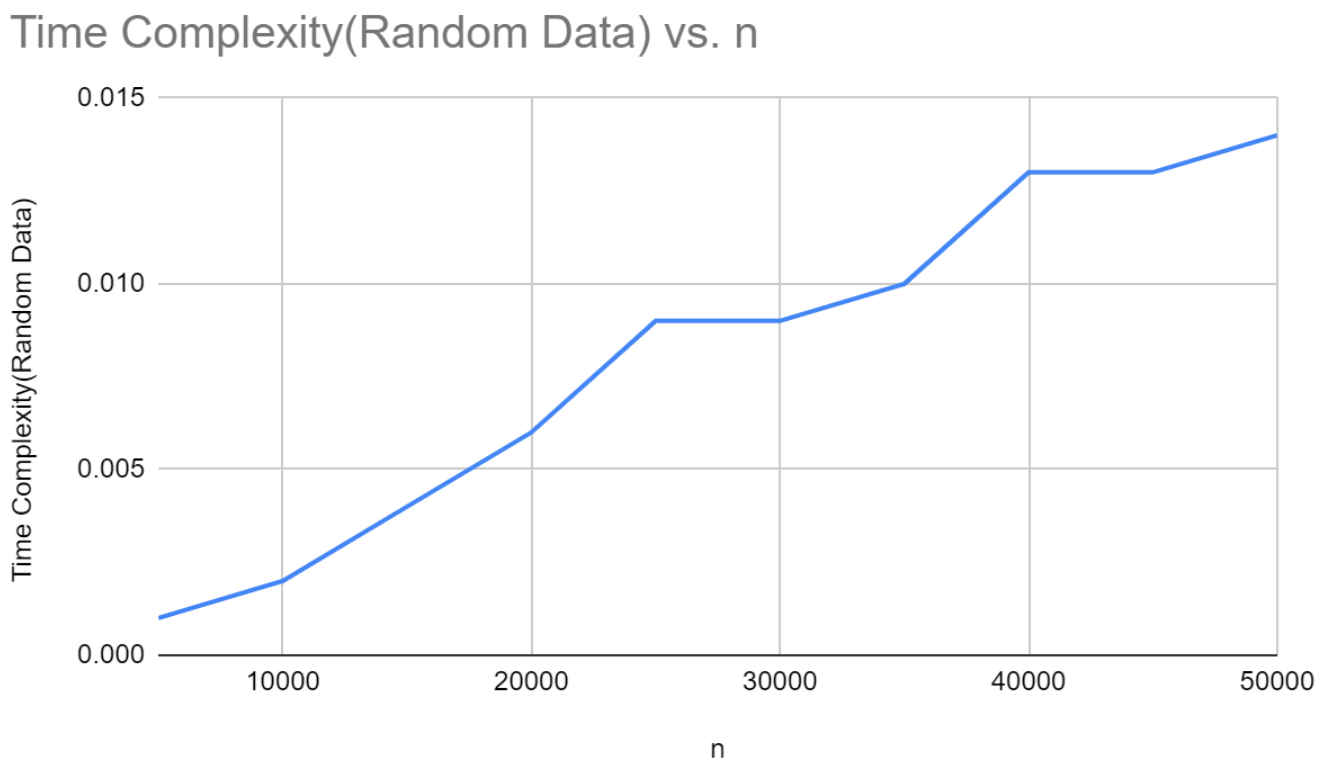
11.
Insert a new element
12.
Delete an element
Enter your choice:

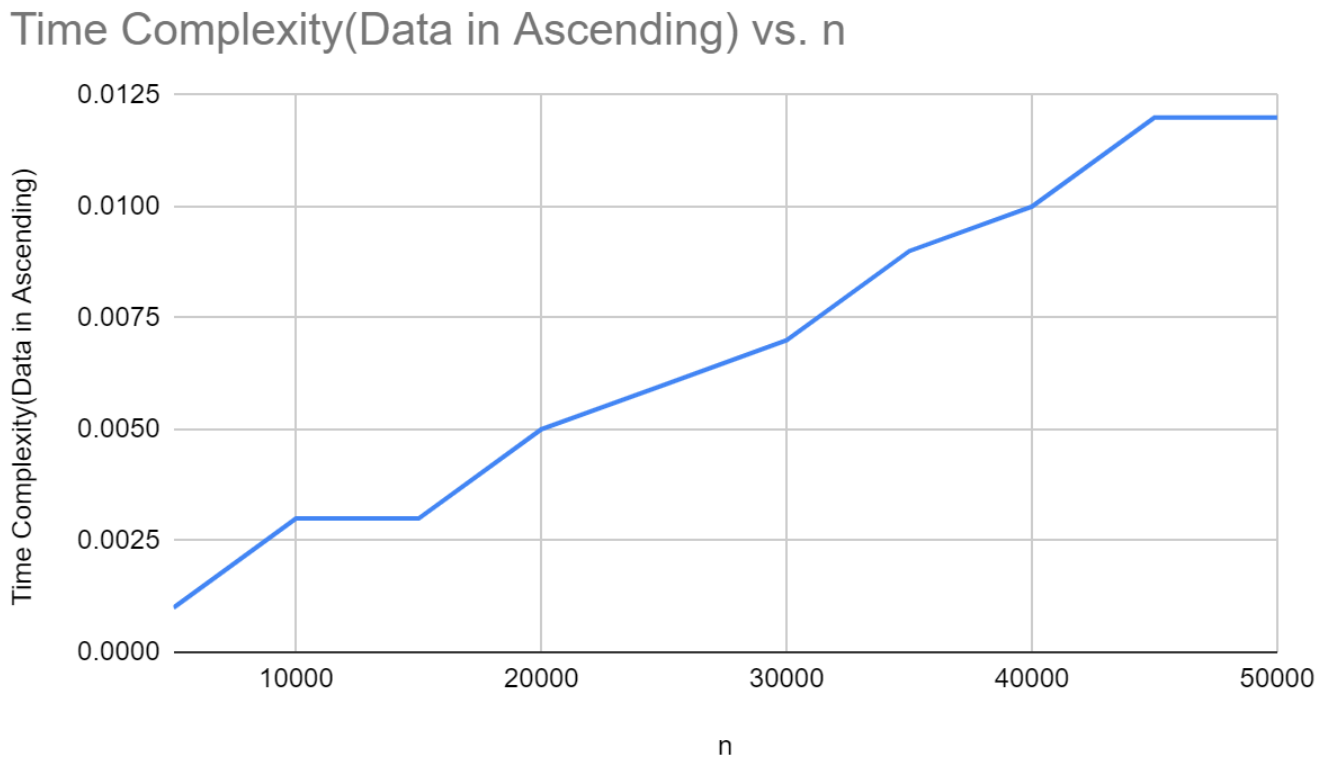
Input/Output Screenshots:

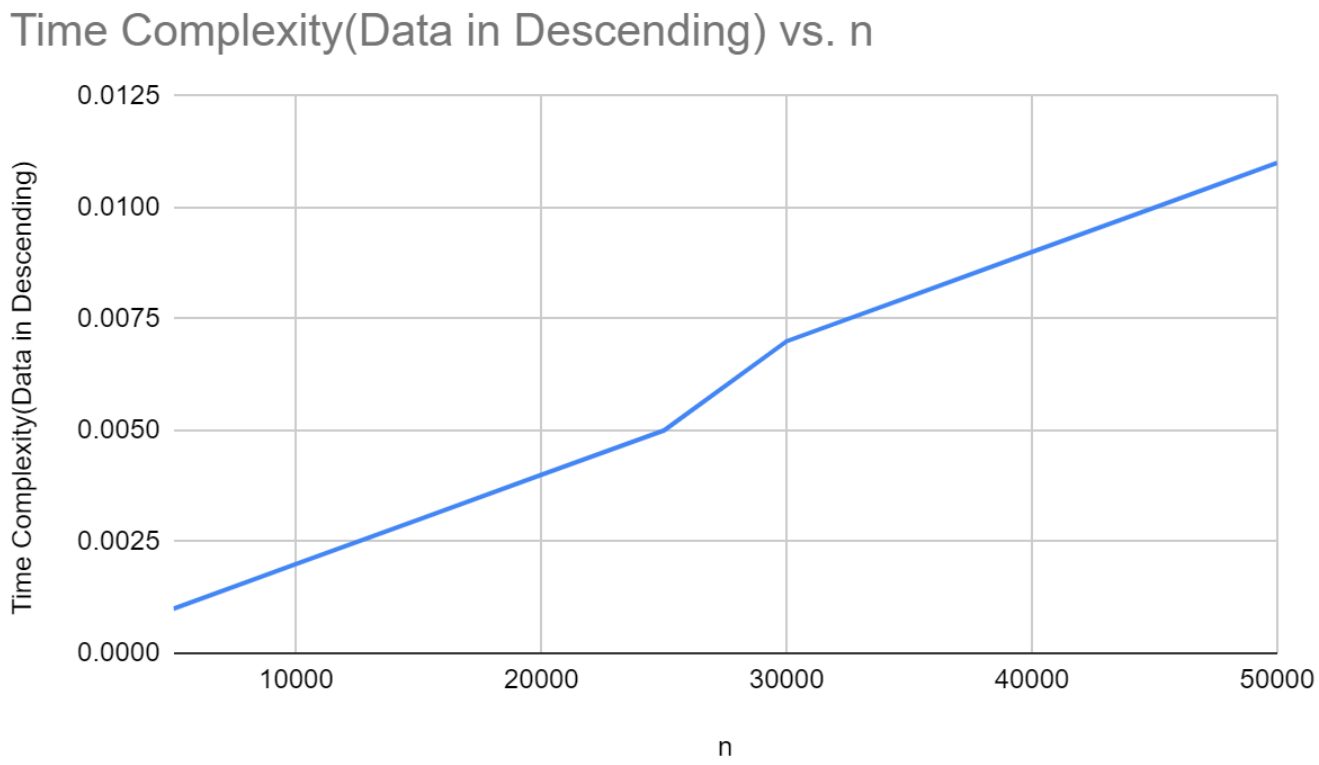
RUN-1:

```
PS C:\Algo-Lab\Lab-5> gcc .\heap_sort_ascending.c
PS C:\Algo-Lab\Lab-5> ./a
0. Quit
1.  n Random numbers=>Array
2.  Display the Array
3.  Sort the Array in Ascending Order by using Max-Heap Sort technique
4.  Sort the Array in Descending Order by using any algorithm
5.  Time Complexity to sort ascending of random data
6.  Time Complexity to sort ascending of data already sorted in ascending order
7.  Time Complexity to sort ascending of data already sorted in descending order
8.  Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000
9.  Extract largest element
10. Replace value at a node with new value
11. Insert a new element
12. Delete an element
8
SI No.  Value of n      Time complexity(Random Data)    Time complexity(Data in Ascending)      Time complexity(Data in Descending)
1       5000            0.001000                        0.001000                                0.001000
2       10000           0.002000                        0.003000                                0.002000
3       15000           0.004000                        0.003000                                0.003000
4       20000           0.006000                        0.005000                                0.004000
5       25000           0.009000                        0.006000                                0.005000
6       30000           0.009000                        0.007000                                0.007000
7       35000           0.010000                        0.009000                                0.008000
8       40000           0.013000                        0.010000                                0.009000
9       45000           0.013000                        0.012000                                0.010000
10      50000           0.014000                        0.012000                                0.011000
0. Quit
1.  n Random numbers=>Array
2.  Display the Array
3.  Sort the Array in Ascending Order by using Max-Heap Sort technique
4.  Sort the Array in Descending Order by using any algorithm
5.  Time Complexity to sort ascending of random data
6.  Time Complexity to sort ascending of data already sorted in ascending order
7.  Time Complexity to sort ascending of data already sorted in descending order
8.  Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000
9.  Extract largest element
10. Replace value at a node with new value
11. Insert a new element
12. Delete an element
0
EXITING
```

The respective graphs are:







Source code

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

void destroy_prev_allocation(int *a, int n) // Destroys previously (dynamically allocated) memory from the array
{
   int i;
   for(i=0; i<n; i++)
   {
      free(a + i);
   }
}

void insert_rand_array(int *a, int n) // Inserts n random values into the array
{
   int i;
   for(i=0; i<n; i++)
   {
      a[i] = rand();
   }
}
```

```
void display(int *a, int n) // Displays array elements
{
   int i;
   for(i=0; i<n; i++)
   {
      printf("%d\t",a[i]);
   }
   printf("\n");
}

int left(int i)
{
   return (2*i + 1);
}

int right(int i)
{
   return (2*i + 2);
}

int parent(int i)
{
   return ((i - 1)/2);
}

void max_heapify(int a[], int n, int i)
{
   int l = left(i);
   int r = right(i);
   int largest = i;
   int temp;

   if(l<n && a[l]>a[largest])
   {
      largest = l;
   }

   if(r<n && a[r]>a[largest])
   {
      largest = r;
   }

   if(largest != i)
   {
      temp = a[i];
      a[i] = a[largest];
      a[largest] = temp;
```

```
        max_heapify(a, n, largest);
    }
}

void build_max_heap(int a[], int n)
{
    int i;
    for(i=n/2 - 1; i>=0; i--)
    {
        max_heapify(a, n, i);
    }
}

void heap_sort_ascending(int a[], int n)
{
    int i, temp;

    build_max_heap(a, n);

    for(i = n-1; i>0; i--)
    {
        temp = a[i];
        a[i] = a[0];
        a[0] = temp;

        max_heapify(a, i, 0);
    }
}

void selection_sort_descending(int *a, int n) // Sorts the array in descending order using selection sort
{
    int i, j, max, temp;
    for(i=0; i<n; i++)
    {
        max = i;
        for(j=i+1; j<n; j++)
        {
            if(a[j] > a[max])
            {
                max = j;
            }
        }

        temp = a[i];
        a[i] = a[max];
        a[max] = temp;
    }
```

```
}

void row_display(int *a, int n)
{
   clock_t start, end;
   double time1, time2, time3;
   insert_rand_array(a, n);

   // Random
   start = clock();
   heap_sort_ascending(a, n);
   end = clock();
   time1 = ((double) (end - start) )/CLOCKS_PER_SEC;

   // Ascending
   start = clock();
   heap_sort_ascending(a, n); //Array is already sorted in ascending order
   end = clock();
   time2 = ((double) (end - start) )/CLOCKS_PER_SEC;

   // Descending
   selection_sort_descending(a, n);
   start = clock();
   heap_sort_ascending(a, n);
   end = clock();
   time3 = ((double) (end - start) )/CLOCKS_PER_SEC;

   printf("%d\t\t%lf\t\t\t%lf\t\t\t\t%lf\n", n, time1, time2, time3);
}

int extract_largest(int a[], int n)
{
   build_max_heap(a, n);

   return a[0];
}

void max_heap_change(int a[], int n, int i, int key)
{
   int temp, p, prev = a[i];
   a[i] = key;
   if(i >= n)
   {
      printf("Invalid index\n");
   }
   else
   {
      if(key < prev)
```

```
        {
            max_heapify(a, n, i);
        }
        else if(key > prev)
        {
            p = parent(i);
            while(i>0 && a[p]<a[i])
            {
                temp = a[i];
                a[i] = a[p];
                a[p] = temp;

                i = p;
                p = parent(i);
            }
        }
    }
}

void insert_at_array_end(int **a, int *n, int key)
{
    int i, temp[*n];

    for(i=0; i<(*n); i++)
    {
        temp[i] = (*a)[i];
    }

    destroy_prev_allocation(*a, *n);

    (*n)++;
    *a = (int*) malloc(sizeof(int) * (*n));

    for(i=0; i<((*n)-1); i++)
    {
        (*a)[i] = temp[i];
    }

    (*a)[i] = key;
}

void max_heap_insert(int **a, int *n, int key)
{
    int i, p, temp;
    insert_at_array_end(a, n, key);

    i = (*n)-1;
    p = parent(i);
```

```
    while(i>0 && (*a)[p]<(*a)[i])
    {
        temp = (*a)[i];
        (*a)[i] = (*a)[p];
        (*a)[p] = temp;

        i = p;
        p = parent(i);
    }
}

void max_heap_delete(int **a, int *n, int i)
{
    int temp = (*a)[i];
    (*a)[i] = (*a)[*n-1];
    (*a)[*n-1] = temp;
    (*n)--;
    free((*a)+(*n));
    build_max_heap(*a, *n);
}

int main()
{
    int choice=1, n=0, *a, i, key, l;
    clock_t start, end;
    double time;

    while(choice)
    {
        printf("0. Quit"
        "\n1.  n Random numbers=>Array"
        "\n2.  Display the Array"
        "\n3.  Sort the Array in Ascending Order by using Max-Heap Sort technique"
        "\n4.  Sort the Array in Descending Order by using any algorithm"
        "\n5.  Time Complexity to sort ascending of random data"
        "\n6.  Time Complexity to sort ascending of data already sorted in ascending order"
        "\n7.  Time Complexity to sort ascending of data already sorted in descending order"
        "\n8.  Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending &
Random Data) in Tabular form for values n=5000 to 50000, step=5000"
        "\n9.  Extract largest element"
        "\n10. Replace value at a node with new value"
        "\n11. Insert a new element"
        "\n12. Delete an element\n");

        scanf("%d", &choice);

        if(choice == 0)
```

```
{
   printf("EXITING\n");
   break;
}
switch(choice)
{
   case 1:
      if(n > 0)
      {
         destroy_prev_allocation(a, n);
      }
      printf("Enter n : ");
      scanf("%d", &n);
      if(n > 0)
      {
         a = (int*) malloc(sizeof(int) * n);
         insert_rand_array(a, n);
      }
      break;

   case 2:
      display(a, n);
      break;

   case 3:
      heap_sort_ascending(a, n);
      display(a, n);
      break;

   case 4:
      selection_sort_descending(a, n);
      display(a,n);
      break;

   case 5:
      if(n > 0)
      {
         destroy_prev_allocation(a,n);
         a = (int*) malloc(sizeof(int) * n);
         insert_rand_array(a, n);
      }
      start = clock();
      heap_sort_ascending(a, n);
      end = clock();
      time = ((double) (end - start) )/CLOCKS_PER_SEC;
      printf("Time taken : %lf seconds\n", time);
   break;
```

```
case 6:
   if(n > 0)
   {
      destroy_prev_allocation(a,n);
      a = (int*) malloc(sizeof(int) * n);
      insert_rand_array(a, n);
   }
   heap_sort_ascending(a, n);
   start = clock();
   heap_sort_ascending(a, n);
   end = clock();
   time = ((double) (end - start) )/CLOCKS_PER_SEC;
   printf("Time taken : %lf seconds\n", time);
   break;

case 7:
   if(n > 0)
   {
      destroy_prev_allocation(a,n);
      a = (int*) malloc(sizeof(int) * n);
      insert_rand_array(a, n);
   }
   selection_sort_descending(a, n);
   start = clock();
   heap_sort_ascending(a, n);
   end = clock();
   time = ((double) (end - start) )/CLOCKS_PER_SEC;
   printf("Time taken : %lf seconds\n", time);
   break;

case 8:
   if(n > 0)
   {
      destroy_prev_allocation(a,n);
      a = (int*) malloc(sizeof(int) * n);
      insert_rand_array(a, n);
   }
   printf("SI No."
   "\tValue of n"
   "\tTime complexity(Random Data)"
   "\tTime complexity(Data in Ascending)"
   "\tTime complexity(Data in Descending)\n"
   );
   for(n=5000; n<=50000; n += 5000)
   {
      a = (int*) malloc( sizeof(int) * n );
      printf("%d\t", (n/5000));
      row_display(a, n);
```

```
            destroy_prev_allocation(a, n);
        }
        break;

    case 9:
        l = extract_largest(a, n);
        printf("Largest element = %d\n", l);
        break;

    case 10:
        build_max_heap(a, n);
        printf("Heap is :\n");
        display(a, n);
        printf("Enter index to replace : ");
        scanf("%d", &i);
        printf("Enter value to replace current value : ");
        scanf("%d",&key);
        max_heap_change(a, n, i, key);
        display(a, n);
        break;

    case 11:
        build_max_heap(a, n);
        printf("Heap is :\n");
        display(a, n);
        printf("Enter value to insert : ");
        scanf("%d", &key);
        max_heap_insert(&a, &n, key);
        display(a, n);
        break;

    case 12:
        build_max_heap(a, n);
        printf("Heap is :\n");
        display(a, n);
        printf("Enter index to delete : ");
        scanf("%d", &i);
        max_heap_delete(&a, &n, i);
        display(a, n);
        break;

    default:
        printf("Invalid choice\n");
    }
  }

  return 0;
}
```

Conclusion/Observation

Thus, we observe that heap sort in ascending order takes almost equal time for ascending, descending and random data. This is probably because heapifying the data and extracting the first element takes identical time for every iteration irrespective of the previous state of the array.

Program No: 5.2

Program Title:

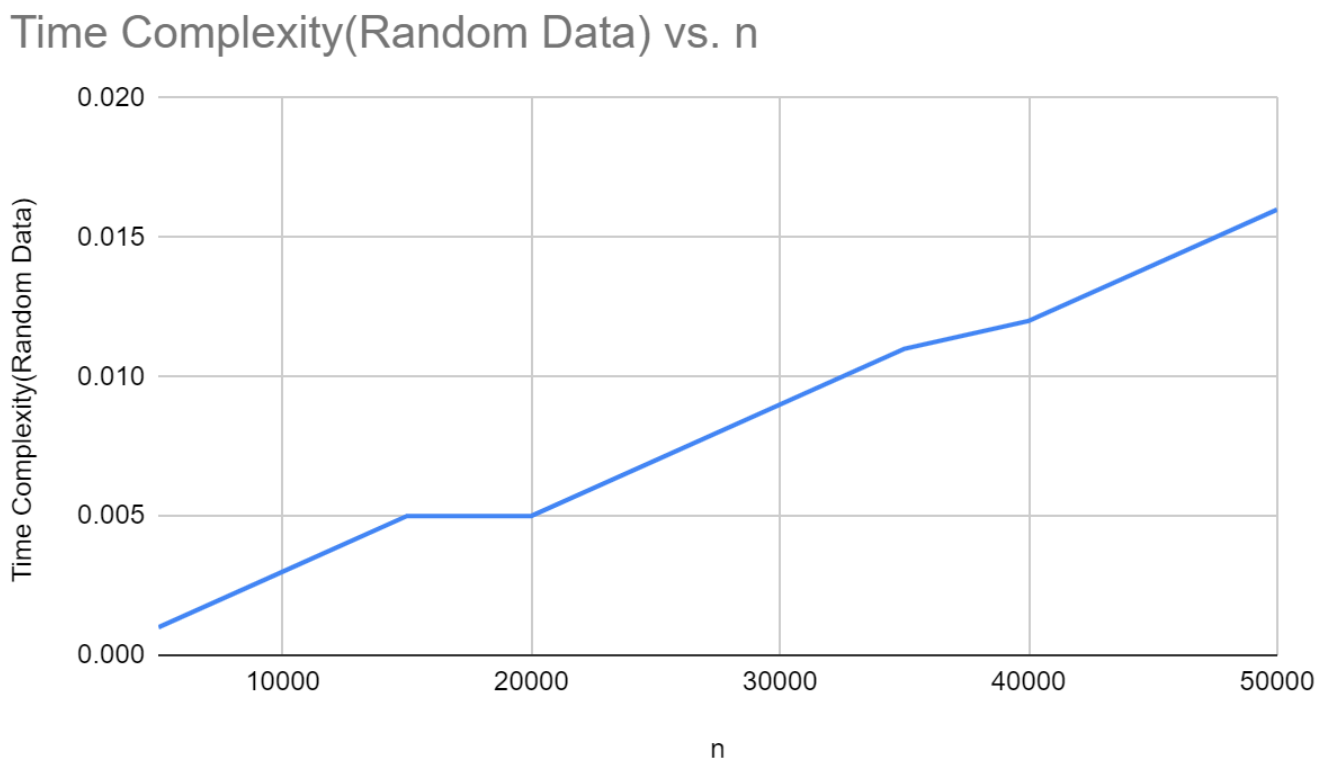
Similar to above program no.5.1, write a menu driven program to sort an array of n integers in descending order by heap sort algorithm. Hints: Use min heap and accordingly change the menu options.

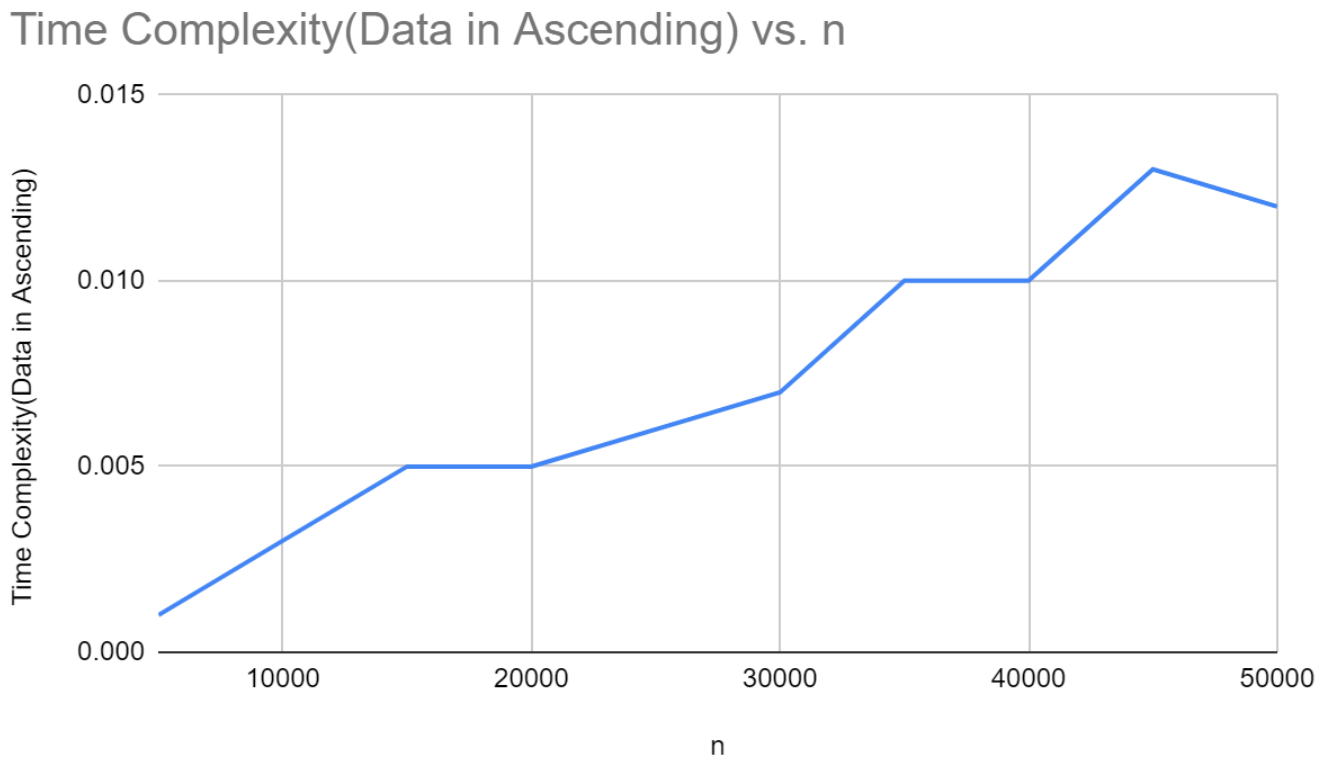
Input/Output Screenshots:

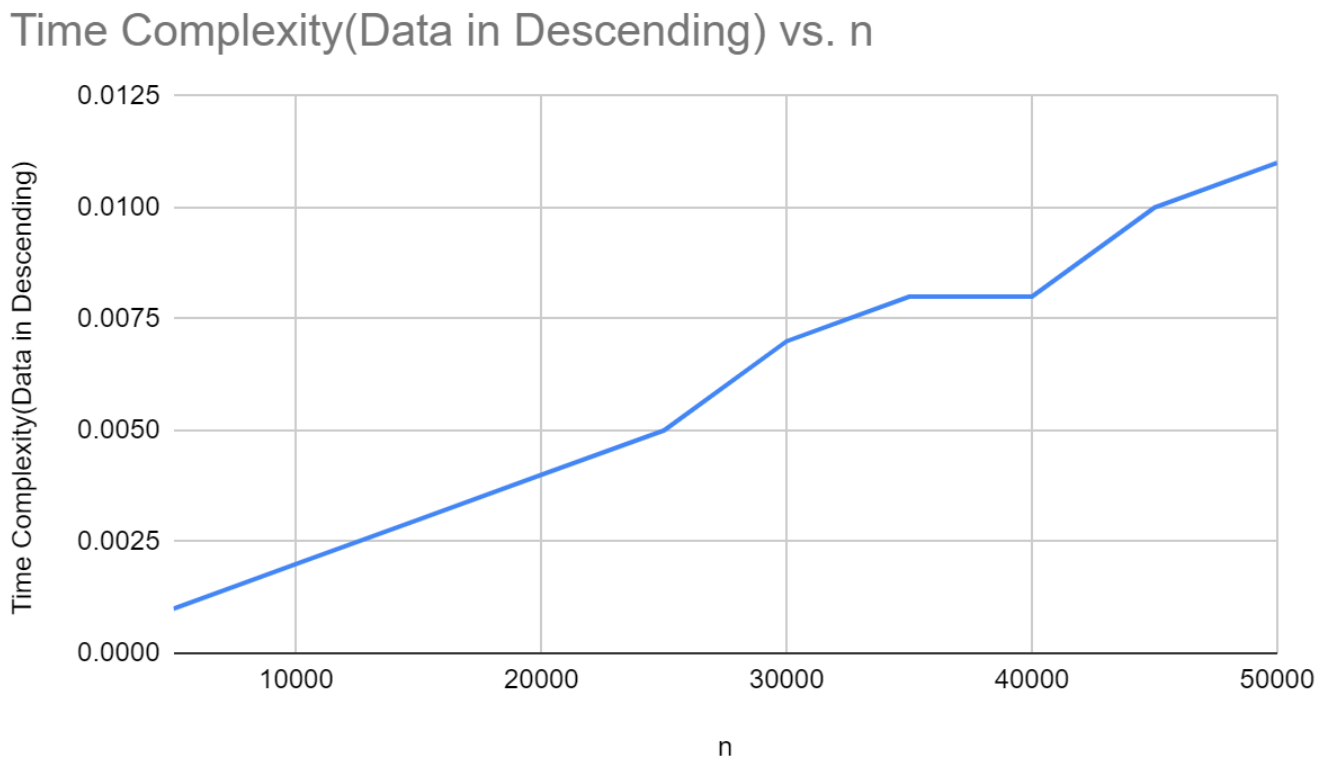
RUN-1:

```
PS C:\Algo-Lab\Lab-5> gcc .\heap_sort_descending.c
PS C:\Algo-Lab\Lab-5> ./a
0. Quit
1.  n Random numbers=>Array
2.  Display the Array
3.  Sort the Array in Descending Order by using Max-Heap Sort technique
4.  Sort the Array in Ascending Order by using any algorithm
5.  Time Complexity to sort descending of random data
6.  Time Complexity to sort descending of data already sorted in ascending order
7.  Time Complexity to sort descending of data already sorted in descending order
8.  Time Complexity to sort descending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000
9.  Extract smallest element
10. Replace value at a node with new value
11. Insert a new element
12. Delete an element
8
SI No.  Value of n      Time complexity(Random Data)    Time complexity(Data in Ascending)      Time complexity(Data in Descending)
1       5000            0.001000                        0.001000                                0.001000
2       10000           0.003000                        0.003000                                0.002000
3       15000           0.005000                        0.005000                                0.003000
4       20000           0.005000                        0.005000                                0.004000
5       25000           0.007000                        0.006000                                0.005000
6       30000           0.009000                        0.007000                                0.007000
7       35000           0.011000                        0.010000                                0.008000
8       40000           0.012000                        0.010000                                0.008000
9       45000           0.014000                        0.013000                                0.010000
10      50000           0.016000                        0.012000                                0.011000
0. Quit
1.  n Random numbers=>Array
2.  Display the Array
3.  Sort the Array in Descending Order by using Max-Heap Sort technique
4.  Sort the Array in Ascending Order by using any algorithm
5.  Time Complexity to sort descending of random data
6.  Time Complexity to sort descending of data already sorted in ascending order
7.  Time Complexity to sort descending of data already sorted in descending order
8.  Time Complexity to sort descending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000
9.  Extract smallest element
10. Replace value at a node with new value
11. Insert a new element
12. Delete an element
0
EXITING
```

The respective graphs are:







Source code

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

void destroy_prev_allocation(int *a, int n) // Destroys previously (dynamically allocated) memory from the array
{
   int i;
   for(i=0; i<n; i++)
   {
      free(a + i);
   }
}

void insert_rand_array(int *a, int n) // Inserts n random values into the array
{
   int i;
   for(i=0; i<n; i++)
   {
      a[i] = rand();
   }
}
```

```
void display(int *a, int n) // Displays array elements
{
   int i;
   for(i=0; i<n; i++)
   {
      printf("%d\t",a[i]);
   }
   printf("\n");
}

int left(int i)
{
   return (2*i + 1);
}

int right(int i)
{
   return (2*i + 2);
}

int parent(int i)
{
   return ((i - 1)/2);
}

void min_heapify(int a[], int n, int i)
{
   int l = left(i);
   int r = right(i);
   int smallest = i;
   int temp;

   if(l<n && a[l]<a[smallest])
   {
      smallest = l;
   }

   if(r<n && a[r]<a[smallest])
   {
      smallest = r;
   }

   if(smallest != i)
   {
      temp = a[i];
      a[i] = a[smallest];
      a[smallest] = temp;
```

```
        min_heapify(a, n, smallest);
    }
}

void build_min_heap(int a[], int n)
{
    int i;
    for(i=n/2 - 1; i>=0; i--)
    {
        min_heapify(a, n, i);
    }
}

void heap_sort_descending(int a[], int n)
{
    int i, temp;

    build_min_heap(a, n);

    for(i = n-1; i>0; i--)
    {
        temp = a[i];
        a[i] = a[0];
        a[0] = temp;

        min_heapify(a, i, 0);
    }
}

void selection_sort_ascending(int *a, int n) // Sorts the array in ascending order using selection sort
{
    int i, j, min, temp;
    for(i=0; i<n; i++)
    {
        min = i;
        for(j=i+1; j<n; j++)
        {
            if(a[j] < a[min])
            {
                min = j;
            }
        }

        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
```

```
}

void row_display(int *a, int n)
{
   clock_t start, end;
   double time1, time2, time3;
   insert_rand_array(a, n);

   // Random
   start = clock();
   heap_sort_descending(a, n);
   end = clock();
   time1 = ((double) (end - start) )/CLOCKS_PER_SEC;

   // Ascending
   start = clock();
   heap_sort_descending(a, n); //Array is already sorted in ascending order
   end = clock();
   time2 = ((double) (end - start) )/CLOCKS_PER_SEC;

   // Descending
   selection_sort_ascending(a, n);
   start = clock();
   heap_sort_descending(a, n);
   end = clock();
   time3 = ((double) (end - start) )/CLOCKS_PER_SEC;

   printf("%d\t\t%lf\t\t\t%lf\t\t\t\t%lf\n", n, time1, time2, time3);
}

int extract_smallest(int a[], int n)
{
   build_min_heap(a, n);

   return a[0];
}

void min_heap_change(int a[], int n, int i, int key)
{
   int temp, p, prev = a[i];
   a[i] = key;
   if(i >= n)
   {
      printf("Invalid index\n");
   }
   else
   {
      if(key > prev)
```

```
        {
            min_heapify(a, n, i);
        }
        else if(key > prev)
        {
            p = parent(i);
            while(i>0 && a[p]>a[i])
            {
                temp = a[i];
                a[i] = a[p];
                a[p] = temp;

                i = p;
                p = parent(i);
            }
        }
    }
}

void insert_at_array_end(int **a, int *n, int key)
{
    int i, temp[*n];

    for(i=0; i<(*n); i++)
    {
        temp[i] = (*a)[i];
    }

    destroy_prev_allocation(*a, *n);

    (*n)++;
    *a = (int*) malloc(sizeof(int) * (*n));

    for(i=0; i<((*n)-1); i++)
    {
        (*a)[i] = temp[i];
    }

    (*a)[i] = key;
}

void min_heap_insert(int **a, int *n, int key)
{
    int i, p, temp;
    insert_at_array_end(a, n, key);

    i = (*n)-1;
    p = parent(i);
```

```
    while(i>0 && (*a)[p]>(*a)[i])
    {
        temp = (*a)[i];
        (*a)[i] = (*a)[p];
        (*a)[p] = temp;

        i = p;
        p = parent(i);
    }
}

void min_heap_delete(int **a, int *n, int i)
{
    int temp = (*a)[i];
    (*a)[i] = (*a)[*n-1];
    (*a)[*n-1] = temp;
    (*n)--;
    free((*a)+(*n));
    build_min_heap(*a, *n);
}

int main()
{
    int choice=1, n=0, *a, i, key, s;
    clock_t start, end;
    double time;

    while(choice)
    {
        printf("0. Quit"
        "\n1.  n Random numbers=>Array"
        "\n2.  Display the Array"
        "\n3.  Sort the Array in Descending Order by using Max-Heap Sort technique"
        "\n4.  Sort the Array in Ascending Order by using any algorithm"
        "\n5.  Time Complexity to sort descending of random data"
        "\n6.  Time Complexity to sort descending of data already sorted in ascending order"
        "\n7.  Time Complexity to sort descending of data already sorted in descending order"
        "\n8.  Time Complexity to sort descending all Cases (Data Ascending, Data in Descending &
Random Data) in Tabular form for values n=5000 to 50000, step=5000"
        "\n9.  Extract smallest element"
        "\n10. Replace value at a node with new value"
        "\n11. Insert a new element"
        "\n12. Delete an element\n");

        scanf("%d", &choice);

        if(choice == 0)
```

```
{
   printf("EXITING\n");
   break;
}
switch(choice)
{
   case 1:
      if(n > 0)
      {
         destroy_prev_allocation(a, n);
      }
      printf("Enter n : ");
      scanf("%d", &n);
      if(n > 0)
      {
         a = (int*) malloc(sizeof(int) * n);
         insert_rand_array(a, n);
      }
      break;

   case 2:
      display(a, n);
      break;

   case 3:
      heap_sort_descending(a, n);
      display(a, n);
      break;

   case 4:
      selection_sort_ascending(a, n);
      display(a,n);
      break;

   case 5:
      if(n > 0)
      {
         destroy_prev_allocation(a,n);
         a = (int*) malloc(sizeof(int) * n);
         insert_rand_array(a, n);
      }
      start = clock();
      heap_sort_descending(a, n);
      end = clock();
      time = ((double) (end - start) )/CLOCKS_PER_SEC;
      printf("Time taken : %lf seconds\n", time);
   break;
```

```
case 6:
   if(n > 0)
   {
      destroy_prev_allocation(a,n);
      a = (int*) malloc(sizeof(int) * n);
      insert_rand_array(a, n);
   }
   heap_sort_descending(a, n);
   start = clock();
   heap_sort_descending(a, n);
   end = clock();
   time = ((double) (end - start) )/CLOCKS_PER_SEC;
   printf("Time taken : %lf seconds\n", time);
   break;

case 7:
   if(n > 0)
   {
      destroy_prev_allocation(a,n);
      a = (int*) malloc(sizeof(int) * n);
      insert_rand_array(a, n);
   }
   selection_sort_ascending(a, n);
   start = clock();
   heap_sort_descending(a, n);
   end = clock();
   time = ((double) (end - start) )/CLOCKS_PER_SEC;
   printf("Time taken : %lf seconds\n", time);
   break;

case 8:
   if(n > 0)
   {
      destroy_prev_allocation(a,n);
      a = (int*) malloc(sizeof(int) * n);
      insert_rand_array(a, n);
   }
   printf("SI No."
   "\tValue of n"
   "\tTime complexity(Random Data)"
   "\tTime complexity(Data in Ascending)"
   "\tTime complexity(Data in Descending)\n"
   );
   for(n=5000; n<=50000; n += 5000)
   {
      a = (int*) malloc( sizeof(int) * n );
      printf("%d\t", (n/5000));
      row_display(a, n);
```

```
            destroy_prev_allocation(a, n);
        }
        break;

    case 9:
        s = extract_smallest(a, n);
        printf("Largest element = %d\n", s);
        break;

    case 10:
        build_min_heap(a, n);
        printf("Heap is :\n");
        display(a, n);
        printf("Enter index to replace : ");
        scanf("%d", &i);
        printf("Enter value to replace current value : ");
        scanf("%d",&key);
        min_heap_change(a, n, i, key);
        display(a, n);
        break;

    case 11:
        build_min_heap(a, n);
        printf("Heap is :\n");
        display(a, n);
        printf("Enter value to insert : ");
        scanf("%d", &key);
        min_heap_insert(&a, &n, key);
        display(a, n);
        break;

    case 12:
        build_min_heap(a, n);
        printf("Heap is :\n");
        display(a, n);
        printf("Enter index to delete : ");
        scanf("%d", &i);
        min_heap_delete(&a, &n, i);
        display(a, n);
        break;

    default:
        printf("Invalid choice\n");
    }
  }

  return 0;
}
```

Conclusion/Observation

Thus, we observe that heap sort in descending order takes almost equal time for ascending, descending and random data. Moreover, it is identical to heap sort in ascending order with respect to time taken. This is probably because heapifying the data and extracting the first element takes identical time for every iteration irrespective of the previous state of the array, and also irrespective of whether we are using max heap or min heap.