**DESIGN AND ANALYSIS OF**

**ALGORITHM LAB (KCS-553)**

**Bachelor of Technology**

in

**Computer Science and Engineering**



**Submitted To: Submitted By:**

MS. SARVESTHA SINGH HARSH SAHU

ASSISTANT PROFESSOR CSE SECTION – 3B

ROLL. NO. 2204220109013

**Bansal Institute Of Engineering and Technology, Lucknow**

**(Dr. A.P.J. Abdul Kalam Technical University, Uttar Pradesh)**

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SR.NO | PROGRAM NAME | PAGE NO. | DATE | TEACHER’S SIGN |
| 01. | PROGRAM FOR RECURSIVE BINARY AND LINERA SEARCH. |  | 08/09/2023 |  |
| 02. | PROGRAM FOR HEAP SORT. |  | 09/09/2023 |  |
| 03. | PROGRAM FOR MERGE SORT. |  | 15/09/2023 |  |
| 04. | PROGRAM FOR SELECTION SORT. |  |  |  |
| 05. | PROGRAM FOR INSERTION SORT. |  |  |  |
| 06. | PROGRAM FOR QUICK SORT. |  |  |  |
| 07. | KNAPSACK PROBLEM USING GREEDY SOLUTION. |  |  |  |
| 08. | FIND THE MINIMUM SPANNING TREE USING KRUSKAL’S ALGORITHM. |  |  |  |
| 09. | PROGRAM FOR KNAPSACK PROBLEM USING DYNAMIC PROGRAMMING. |  |  |  |
|  |  |  |  |  |

**PROGRAM: 01**

**OBJECTIVE:** Implement Recursive Binary search and linear search and determine the time taken to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int linearSearch(int arr[], int size, int key, int currentIndex);

int binarySearch(int arr[], int key, int low, int high);

void input(int arr[], int size);

void delay(int seconds);

int main()

{

int choice, size, key, result;

clock\_t start\_time, end\_time;

printf("\nEnter choice: 1. Linear Search 2. Binary Search: ");

scanf("%d", &choice);

printf("\nEnter size of array: ");

scanf("%d", &size);

int arr[size];

input(arr, size);

printf("\nEnter element to be searched: ");

scanf("%d", &key);

start\_time = clock();

switch (choice)

{

case 1:

result = linearSearch(arr, size, key, 0);

break;

case 2:

result = binarySearch(arr, key, 0, size - 1);

break;

default:

printf("\nWrong choice!");

exit(1);

}

end\_time = clock();

if (result != -1)

printf("\n%d is found at position: %d", key, (result + 1));

else

printf("\n%d is not found in arr!", key);

printf("\nTime consumed: %ld ms\n", (end\_time - start\_time));

return 0;

}

int binarySearch(int arr[], int key, int low, int high)

{

if (low > high)

return -1;

int mid = (low + high) / 2;

delay(1);

if (arr[mid] == key)

return mid;

if (arr[mid] > key)

return binarySearch(arr, key, low, mid - 1);

if (arr[mid] < key)

return binarySearch(arr, key, mid + 1, high);

}

int linearSearch(int arr[], int size, int key, int currentIndex)

{

delay(1);

if (currentIndex == size)

return -1;

if (arr[currentIndex] == key)

return currentIndex;

return linearSearch(arr, size, key, currentIndex + 1);

}

void input(int arr[], int size)

{

for (int i = 0; i < size; i++)

{

printf("Enter element no. %d: ", i + 1);

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

}

}

void delay(int seconds)

{

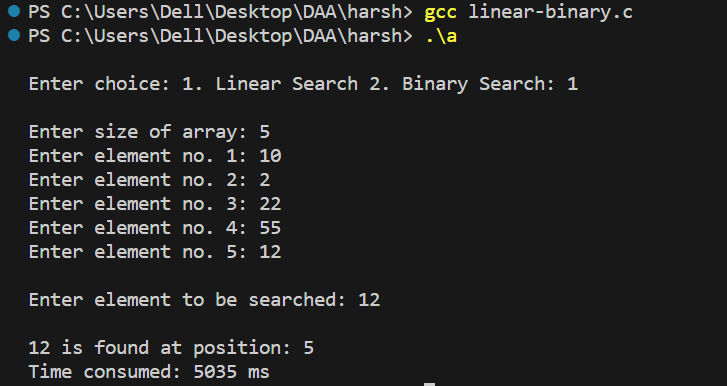
int milliseconds = 1000 \* seconds;

clock\_t start\_time = clock();

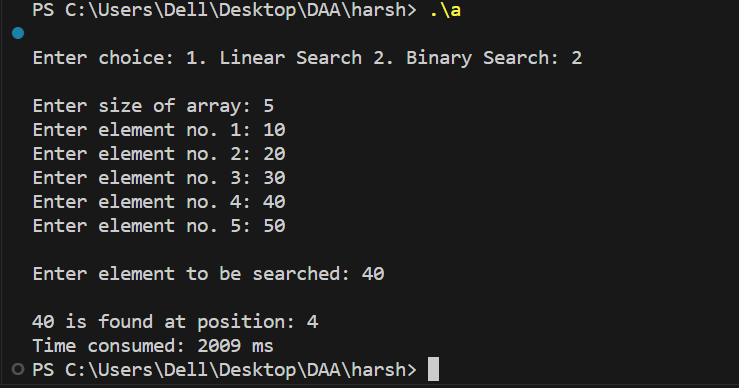
while (clock() < start\_time + milliseconds);

}

**OUTPUT:**



Linear Search



Binary Search

**PROGRAM: 02**

**OBJECTIVE:** Sort a given set of elements using the Heap sort method and determine the time taken to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

**PROGRAM:**

#include <stdio.h>

#include <time.h>

void heapSort(int arr[], int size);

void heapify(int arr[], int size, int i);

void delay(int seconds);

void swap(int \*a, int \*b);

void input(int arr[], int size);

void traverse(int arr[], int size);

int main()

{

int size;

clock\_t start\_time, end\_time;

printf("\nEnter size of array: ");

scanf("%d", &size);

int arr[size];

input(arr, size);

printf("\nInitial Array: ");

traverse(arr, size);

start\_time = clock();

heapSort(arr, size);

end\_time = clock();

printf("\nSorted Array: ");

traverse(arr, size);

printf("\nTime consumed: %ld ms\n", end\_time - start\_time);

return 0;

}

void heapSort(int arr[], int size)

{

delay(1);

for (int i = size / 2 - 1; i >= 0; i--)

heapify(arr, size, i);

for (int i = size - 1; i >= 0; i--)

{

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

void heapify(int arr[], int size, int i)

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < size && arr[left] > arr[largest])

largest = left;

if (right < size && arr[right] > arr[largest])

largest = right;

if (largest != i)

{

swap(&arr[i], &arr[largest]);

heapify(arr, size, largest);

}

}

void input(int arr[], int size)

{

for (int i = 0; i < size; i++)

{

printf("Enter element no. %d: ", i + 1);

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

}

}

void delay(int seconds)

{

int milliseconds = 1000 \* seconds;

clock\_t start\_time = clock();

while (clock() < start\_time + milliseconds);

}

void traverse(int arr[], int size)

{

for (int i = 0; i < size; i++)

printf("%d ", arr[i]);

}

void swap(int \*a, int \*b)

{

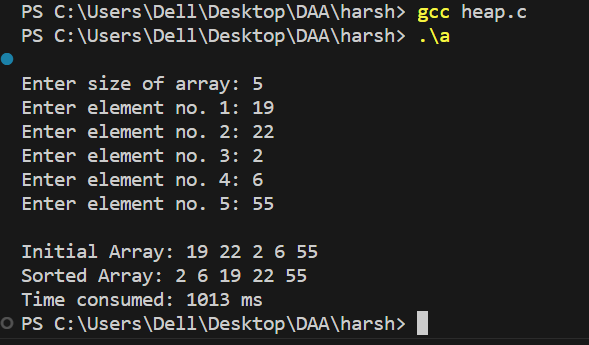
int temp = \*a;

\*a = \*b;

\*b = temp;

}

**OUTPUT:**



**PROGRAM: 03**

**OBJECTIVE:** Sort a given set of elements using Merge sort method and determine the time taken to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

**PROGRAM:**

#include <stdio.h>

#include <time.h>

void mergeSort(int arr[], int left, int right);

void merge(int arr[], int left, int mid, int right);

void traverse(int[], int);

void input(int arr[], int size);

void delay(int seconds);

int main()

{

int size;

clock\_t start\_time, end\_time;

printf("\nEnter size of array: ");

scanf("%d", &size);

int arr[size];

input(arr, size);

printf("\nInitial Array: ");

traverse(arr, size);

start\_time = clock();

mergeSort(arr, 0, size - 1);

end\_time = clock();

printf("\nSorted Array: ");

traverse(arr, size);

printf("\nTime consumed: %ld ms\n", end\_time - start\_time);

return 0;

}

void mergeSort(int arr[], int left, int right)

{

if (left < right)

{

delay(1);

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

void merge(int arr[], int left, int mid, int right)

{

int i, j, k;

int n1 = mid - left + 1;

int n2 = right - mid;

int L[n1], R[n2];

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

L[i] = arr[left + i];

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

R[j] = arr[mid + 1 + j];

i = 0;

j = 0;

k = left;

while (i < n1 && j < n2)

{

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

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void traverse(int arr[], int size)

{

for (int i = 0; i < size; i++)

printf("%d ", arr[i]);

}

void input(int arr[], int size)

{

for (int i = 0; i < size; i++)

{

printf("Enter element no. %d: ", i + 1);

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

}

}

void delay(int seconds)

{

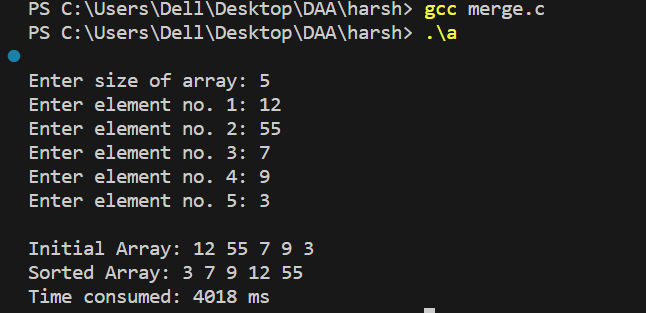
int milliseconds = 1000 \* seconds;

clock\_t start\_time = clock();

while (clock() < start\_time + milliseconds);

}

**OUTPUT:**



**PROGRAM: 04**

**OBJECTIVE:** Sort a given set of elements using Selection sort and hence find the time required to sort elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

**PROGRAM:**

#include <stdio.h>

#include <time.h>

void selectionSort(int[], int);

void delay(int seconds);

void swap(int \*, int \*);

void traverse(int[], int);

void input(int arr[], int size);

int main()

{

int size;

clock\_t start\_time, end\_time;

printf("\nEnter size of array: ");

scanf("%d", &size);

int arr[size];

input(arr, size);

printf("\nInitial Array: ");

traverse(arr, size);

start\_time = clock();

selectionSort(arr, size);

end\_time = clock();

printf("\nSorted Array: ");

traverse(arr, size);

printf("\nTime consumed: %ld ms\n", end\_time - start\_time);

}

void selectionSort(int arr[], int size)

{

delay(1);

for (int i = 0; i < size - 1; i++)

for (int j = i + 1; j < size; j++)

if (arr[i] > arr[j])

swap(&arr[i], &arr[j]);

}

void delay(int seconds)

{

int milliseconds = 1000 \* seconds;

clock\_t start\_time = clock();

while (clock() < start\_time + milliseconds);

}

void swap(int \*num1, int \*num2)

{

int temp = \*num1;

\*num1 = \*num2;

\*num2 = temp;

}

void traverse(int arr[], int size)

{

for (int i = 0; i < size; i++)

printf("%d ", arr[i]);

}

void input(int arr[], int size)

{

for (int i = 0; i < size; i++)

{

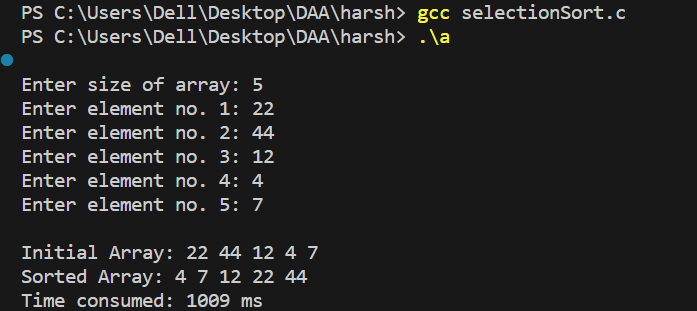
printf("Enter element no. %d: ", i + 1);

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

}

}

**OUTPUT:**



**PROGRAM: 05**

**OBJECTIVE:** Sort a given set of elements using Insertion sort.

**PROGRAM:**

#include <stdio.h>

#include <time.h>

void insertionSort(int arr[], int size);

void delay(int seconds);

void traverse(int[], int);

void input(int arr[], int size);

int main()

{

int size;

clock\_t start\_time, end\_time;

printf("\nEnter size of array: ");

scanf("%d", &size);

int arr[size];

input(arr, size);

printf("\nInitial Array: ");

traverse(arr, size);

start\_time = clock();

insertionSort(arr, size);

end\_time = clock();

printf("\nSorted Array: ");

traverse(arr, size);

printf("\nTime consumed: %ld ms\n", end\_time - start\_time);

}

void insertionSort(int arr[], int size)

{

delay(1);

for (int i = 1; i < size; i++)

{

int hole = i, temp = arr[i];

while (hole > 0 && arr[hole - 1] > temp)

{

arr[hole] = arr[hole - 1];

hole--;

}

arr[hole] = temp;

}

}

void delay(int seconds)

{

int milliseconds = 1000 \* seconds;

clock\_t start\_time = clock();

while (clock() < start\_time + milliseconds)

;

}

void traverse(int arr[], int size)

{

for (int i = 0; i < size; i++)

printf("%d ", arr[i]);

}

void input(int arr[], int size)

{

for (int i = 0; i < size; i++)

{

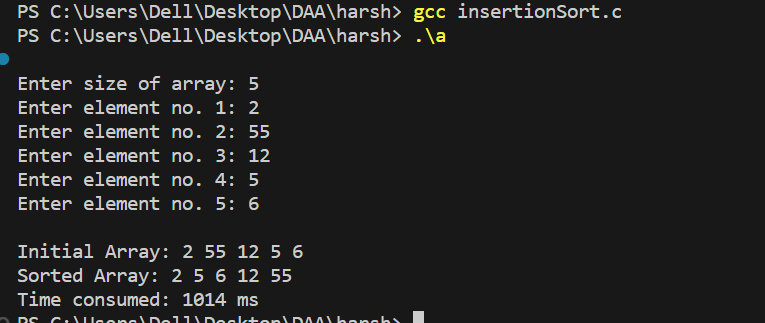
printf("Enter element no. %d: ", i + 1);

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

}

}

**OUTPUT:**



**PROGRAM: 06**

**OBJECTIVE:** Sort a given set of elements using Quick sort method and determine the time taken to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n.

**PROGRAM:**

#include <stdio.h>

#include <time.h>

void quickSort(int[], int, int);

int partition(int[], int, int);

void swap(int \*, int \*);

void traverse(int[], int);

void input(int arr[], int size);

void delay(int seconds);

int main()

{

int size;

clock\_t start\_time, end\_time;

printf("\nEnter size of array: ");

scanf("%d", &size);

int arr[size];

input(arr, size);

printf("\nInitial Array: ");

traverse(arr, size);

start\_time = clock();

quickSort(arr, 0, size - 1);

end\_time = clock();

printf("\nSorted Array: ");

traverse(arr, size);

printf("\nTime consumed: %ld ms\n", end\_time - start\_time);

return 0;

}

void quickSort(int arr[], int start, int end)

{

if (start < end)

{

delay(1);

int splitIndex = partition(arr, start, end);

quickSort(arr, start, splitIndex - 1);

quickSort(arr, splitIndex + 1, end);

}

}

int partition(int arr[], int start, int end)

{

int pivot = arr[end];

int pIndex = start;

for (int i = start; i < end; i++)

{

if (arr[i] <= pivot)

{

swap(&arr[i], &arr[pIndex]);

++pIndex;

}

}

swap(&arr[pIndex], &arr[end]);

return pIndex;

}

void swap(int \*num1, int \*num2)

{

int temp = \*num1;

\*num1 = \*num2;

\*num2 = temp;

}

void traverse(int arr[], int size)

{

for (int i = 0; i < size; i++)

printf("%d ", arr[i]);

}

void input(int arr[], int size)

{

for (int i = 0; i < size; i++)

{

printf("Enter element no. %d: ", i + 1);

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

}

}

void delay(int seconds)

{

int milliseconds = 1000 \* seconds;

clock\_t start\_time = clock();

while (clock() < start\_time + milliseconds);

}

**OUTPUT:**

