

Lab No: 2

Date: 2080/11/27

A. Write a program in C implementing Binary Search.

Theory:

Binary Search is an efficient searching algorithm that works on a sorted array by repeatedly dividing the search space in half.

Working Principle:

- Compare the target element with the middle element.
- If it matches, return its index.
- If smaller, search in the left half; if larger, search in the right half.
- Repeat until the element is found or the search space becomes empty.

Time Complexity: $O(\log n)$

Source Code:

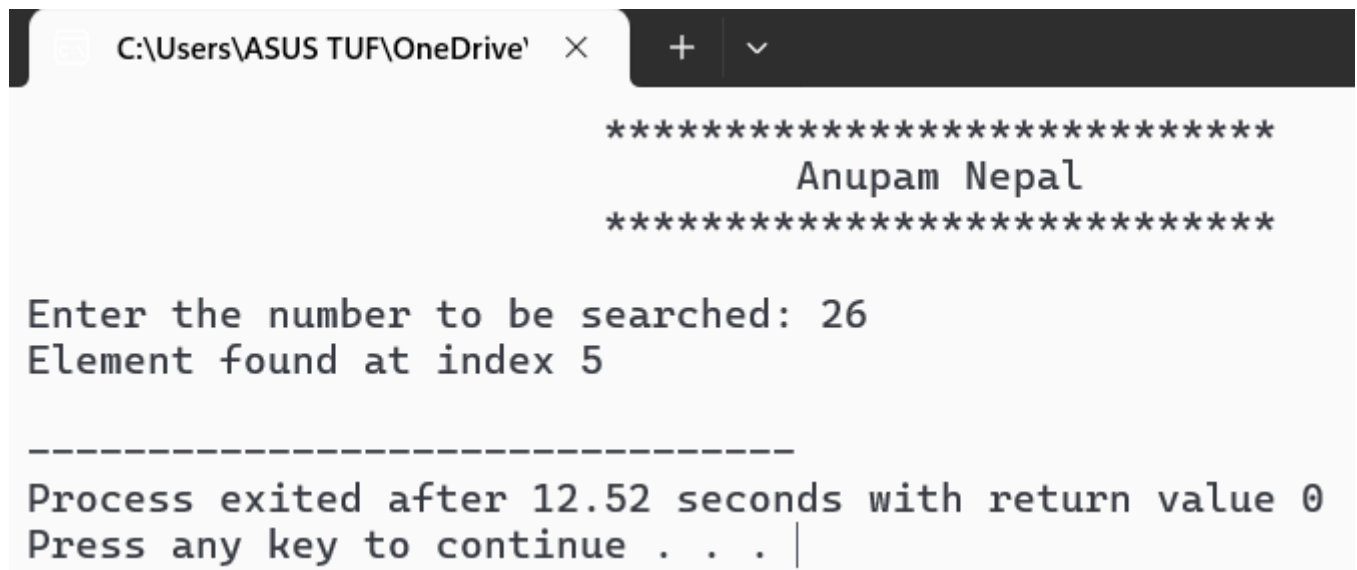
```
#include <stdio.h>

int binary_search(int arr[], int n, int target) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target)
            return mid;
        else if (arr[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}

int main() {
    int arr[] = {1, 3, 8, 12, 18, 26, 33, 46, 56, 77};
```

```
int n = sizeof(arr) / sizeof(arr[0]);
int target,i;
printf("Enter the number to be searched: ");
scanf("%d",&target);
int result = binary_search(arr, n, target);
if (result != -1)
    printf("Element found at index %d\n", result);
else
    printf("Element not found\n");
return 0;
}
```

OUTPUT:



```
C:\Users\ASUS TUF\OneDrive' × + v

*****
Anupam Nepal
*****

Enter the number to be searched: 26
Element found at index 5

-----
Process exited after 12.52 seconds with return value 0
Press any key to continue . . . |
```

B. Write a program in C implementing Min-Max Algorithm.

Theory:

The **Min-Max Problem** is finding the minimum and maximum values in an array.

- **Naïve Approach:** Traverse the array once to find the minimum and again to find the maximum.
- **Optimized Approach:** Traverse the array once, keeping track of both min and max simultaneously.

The Minimax algorithm is a decision-making algorithm used in game theory and artificial intelligence to determine the best move for a player, assuming the opponent also plays optimally.

It is commonly applied in two-player turn-based games like chess, tic-tac-toe, and checkers. The algorithm works by exploring all possible moves and their outcomes, evaluating each position to decide the best move for the maximizing player while the minimizing player aims to counteract this.

Source Code:

```
#include <stdio.h>

void find_min_max(int arr[], int n, int *min, int *max)

{

    int i;

    *min = *max = arr[0];

    for (i = 1; i < n; i++) {

        if (arr[i] < *min)

            *min = arr[i];

        if (arr[i] > *max)

            *max = arr[i];

    }

}

int main()

{

    int arr[] = {5, 13, 45, 66, 22, 18, 88, 4, 3};

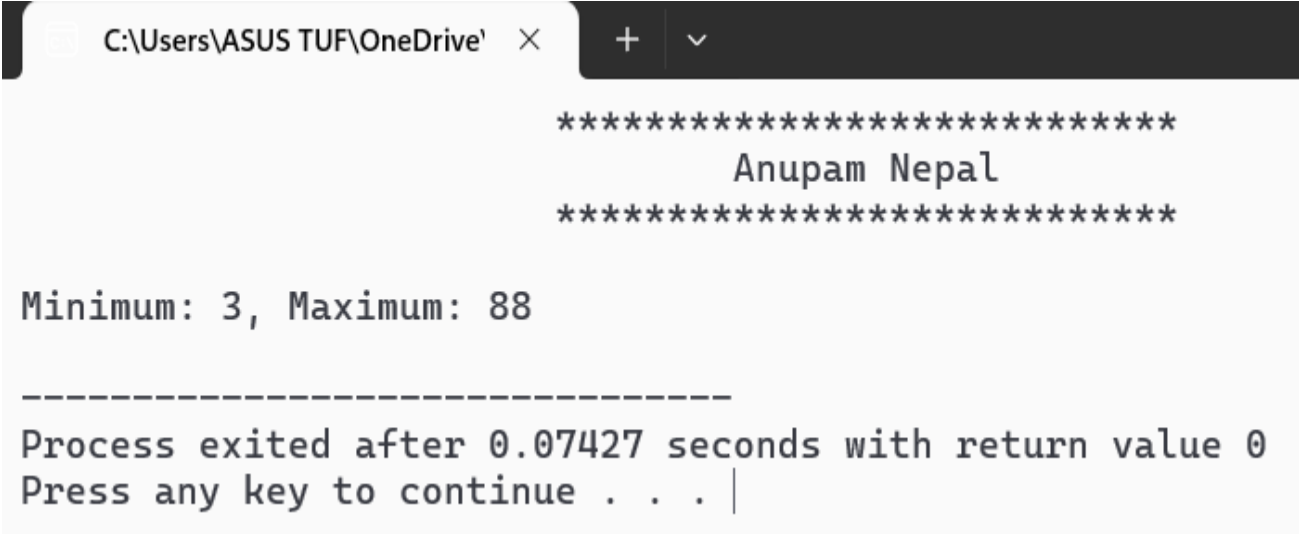
    int n = sizeof(arr) / sizeof(arr[0]);

    int min, max;

    find_min_max(arr, n, &min, &max);
```

```
printf("Minimum: %d, Maximum: %d\n", min, max);  
  
return 0;  
}
```

OUTPUT:



The screenshot shows a Windows command prompt window with a single tab titled 'C:\Users\ASUS TUF\OneDrive'. The window contains the following text:

```
*****  
Anupam Nepal  
*****  
  
Minimum: 3, Maximum: 88  
  
-----  
Process exited after 0.07427 seconds with return value 0  
Press any key to continue . . . |
```

C. Write a program in C implementing Merge Sort.

Theory:

Merge Sort is a divide-and-conquer sorting algorithm that recursively splits the array into halves, sorts them, and merges the sorted halves. It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the sorted array.

Steps:

1. Divide the array into two halves.
2. Recursively sort both halves.
3. Merge the two sorted halves into one sorted array.

Time Complexity: $O(n \log n)$

Source Code:

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right)
{
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int i, j, k, 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++];
        else
            arr[k++] = R[j++];
    }
    while (i < n1)
```

```

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

while (j < n2)

    arr[k++] = R[j++];

}

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

{

    if (left < right) {

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

        merge_sort(arr, left, mid);

        merge_sort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

void print_array(int arr[], int n)

{

    int i;

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

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

    printf("\n");

}

int main() {

    int arr[] = { 12, 31, 23, 5, 46, 17, 84, 25};

    int n = sizeof(arr) / sizeof(arr[0]);

    merge_sort(arr, 0, n - 1);

    printf("Sorted array: ");

    print_array(arr, n);

    return 0;

}

```

OUTPUT:

```
C:\Users\ASUS TUF\OneDrive' X + v
*****
Anupam Nepal
*****

Sorted array: 5 12 17 23 25 31 46 84

-----
Process exited after 0.07417 seconds with return value 0
Press any key to continue . . . |
```

D. Write a program in C implementing Quick Sort.

Theory:

Quick Sort is a divide-and-conquer sorting algorithm that selects a **pivot** and partitions the array into elements smaller and larger than the pivot.

Steps:

1. Choose a **pivot** element.
2. Partition the array around the pivot.
3. Recursively apply Quick Sort to left and right partitions.

Time Complexity: $O(n^2)$

Source Code:

```
#include <stdio.h>

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int partition(int arr[], int low, int high)
{
    int i, j;
    int pivot = arr[high];
    i = (low - 1);
    for (j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i+1], &arr[high]);
    return i+1;
}
```



```

    }

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quick_sort(int arr[], int low, int high)

{
    if (low < high) {

        int pi = partition(arr, low, high);

        quick_sort(arr, low, pi - 1);

        quick_sort(arr, pi + 1, high);

    }

}

void print_array(int arr[], int n)

{

    int i;

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

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

    printf("\n");

}

int main()

{

    int arr[] = {5, 17, 8, 29, 11, 55, 36, 15};

    int n = sizeof(arr) / sizeof(arr[0]);

    quick_sort(arr, 0, n - 1);

    printf("Sorted array: ");

    print_array(arr, n);

    return 0;

}

```

OUTPUT:

```
C:\Users\ASUS TUF\OneDrive' X + v
*****
      Anupam Nepal
*****

Sorted array: 5 8 11 15 17 29 36 55

-----
Process exited after 0.07694 seconds with return value 0
Press any key to continue . . . |
```