Weekly Homework 2

March 26, 2025

1 Linear search

A simple algorithm that sequentially checks each element of the array until the desired element (k) is found or the list ends.

```
int linearSearch(int a[], int n, int k) {
    for (int i = 0; i < n; i++) {
        if (a[i] == k) {
            return i;  // Return the index directly
        }
    }
    return -1; // Return -1 if not found
}
```

2 Linear search with sentinel

A slight optimization where we place a "sentinel" at the end of the array to eliminate the need for a boundary check inside the loop.

```
int sentinelLinearSearch(int a[], int n, int k) {
    int last = a[n - 1];  // Store the last element
    a[n - 1] = k;  // Set the sentinel

    int i = 0;
    while (a[i] != k) {
        i++;
    }

    a[n - 1] = last;  // Restore the last element

    // If found within original range, return index; otherwise, return -1
    if (i < n - 1 || a[n - 1] == k)
        return i;

```

```
    return -1;
}
```

3 Binary search

Binary search is a search algorithm used to find the position of a target value within a sorted array. It works by repeatedly dividing the search interval in half until the target value is found or the interval is empty. The search interval is halved by comparing the target element with the middle value of the search space.

```
// An iterative binary search function.
int binarySearch(int arr[], int low, int high, int x)
{
    while (low <= high) {
        int mid = low + (high - low) / 2;

        // Check if x is present at mid
        if (arr[mid] == x)
            return mid;

        // If x greater, ignore left half
        if (arr[mid] < x)
            low = mid + 1;

        // If x is smaller, ignore right half
        else
            high = mid - 1;
    }

    // If we reach here, then element was not present
    return -1;
}
```

3.1 Binary Search on a Condition

Instead of searching for a value in a sorted array, we search for the smallest value that makes a condition true.

We define a function $f(\text{mid})$ that returns:

- **False** when mid is too small
- **True** when mid is large enough (or meets the condition)

This function must be **monotonic** — once it becomes **True**, it stays **True**.

4 Randomized search

```cpp
#include <iostream>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For seeding rand()

using namespace std;

int randomizedSearch(int a[], int n, int k)
{
    srand(time(0)); // Seed for random number generation
    int count = 0, i;

    // Set an upper limit for iterations to avoid infinite loops
    int maxAttempts = pow(n, 0.9);

    while (count < maxAttempts)
    {
        i = rand() % n; // Generate a random index in range [0, n-1]

        if (a[i] == k)
            return i; // Return index if key is found

        count++;
    }

    return -1; // Return -1 if element is not found
}

```

5 Exercises

Exercise 1

Given an array of N integers and a target integer K, implement a program to find the first occurrence of K in the array using the **Linear Search** algorithm. If K is found, return its index (0-based). Otherwise, return -1.

Example

Input

```
5
1 3 5 7 9
5
```

Output

```
2
```

Exercise 2

Use **Linear Search with Sentinel** to solve Exercise 1.

Exercise 3

Suppose an array of length n sorted in ascending order is **rotated** between 1 and n times. For example, the array

$$\text{nums} = [0, 1, 2, 4, 5, 6, 7]$$

might become:

- $[4, 5, 6, 7, 0, 1, 2]$ if it was rotated 4 times.
- $[0, 1, 2, 4, 5, 6, 7]$ if it was rotated 7 times.

Notice that rotating an array

$$[a_0, a_1, a_2, \ldots, a_{n-1}]$$

one time results in the array

$$[a_{n-1}, a_0, a_1, a_2, \ldots, a_{n-2}].$$

Given the sorted rotated array `nums` of **unique** elements, return *the minimum element of this array.*

Your algorithm must run in $O(\log n)$ time.

Example

Input

```
5
3 4 5 1 2
```

Output

```
1
```

Explanation: The original array was $[1, 2, 3, 4, 5]$ rotated 3 times.

Exercise 4

A conveyor belt has packages that must be shipped from one port to another within `days` days.

The i^{th} package on the conveyor belt has a weight of `weights[i]`. Each day, we load the ship with packages on the conveyor belt (in the order given by `weights`). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within `days` days.

Example

Input

```
10
1 2 3 4 5 6 7 8 9 10
5 # days = 5
```

Output

```
15
```

Explanation:

```
A ship capacity of 15 is the minimum to ship all the packages in 5 days like
1st day: 1, 2, 3, 4, 5
2nd day: 6, 7
3rd day: 8
4th day: 9
5th day: 10
```

Excercise 5

Given an array of positive integers `nums` and a positive integer `target`, return the *minimal length* of a **subarray** whose sum is greater than or equal to `target`. If there is no such subarray, return 0 instead.

Example

Input:

```
target = 7, nums = [2,3,1,2,4,3]
```

Output:

```
2
```

Explanation: The subarray $[4, 3]$ has the minimal length under the problem constraint.

Exercise 6

Given a sorted array of integers and a target sum, determine if there exist two numbers in the array whose sum equals the target.

Example

Input:

```
5
1 2 3 4 6
5
```

Output:

```
YES
```

Exercise 7

Given an integer array `nums`, return all the triplets $[\texttt{nums}[i], \texttt{nums}[j], \texttt{nums}[k]]$ where:

$$\texttt{nums}[i] + \texttt{nums}[j] + \texttt{nums}[k] = 0$$

and the indices i, j, and k are all distinct.

The output should **not** contain any duplicate triplets. You may return the output and the triplets in **any order**.

Example

Input:

```
nums = [-1,0,1,2,-1,-4]
```

Output:

```
[[-1,-1,2], [-1,0,1]]
```

Explanation: The triplets $[-1, -1, 2]$ and $[-1, 0, 1]$ satisfy the condition where their sum equals zero. No duplicate triplets are allowed.

Submission Rules

Students must adhere to the following submission guidelines:

1. Each solution must be submitted in a separate file:
    - `ex1.cpp` for the Fibonacci Exercise 1.
    - `ex2.cpp` for the Factorial Exercise 2.

2. The program should read input from **standard input** (`cin`) and output results to **standard output** (`cout`).

3. Code should be well-structured with proper indentation and comments explaining the logic.

4. The submission must be in a **compressed zip file** named **MSSV.zip**, containing:

   - The required C++ files. (ex1.cpp, ex2.cpp, ex3.cpp, ex4.cpp, ex5.cpp, etc.).
   - A `report.pdf` file describing the approach used in each solution.

5. Example Input/Output Format:

   - **Input:**

     ```
     5
     ```

   - **Output for Fibonacci:**

     ```
     0 1 1 2 3
     ```