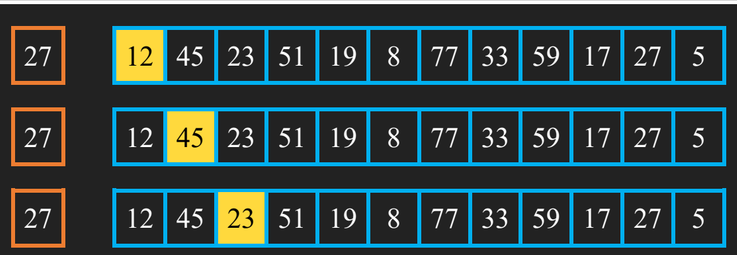
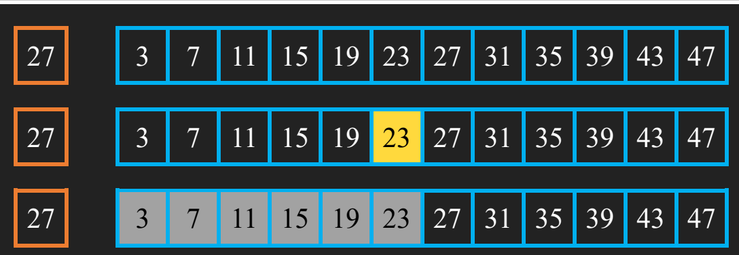
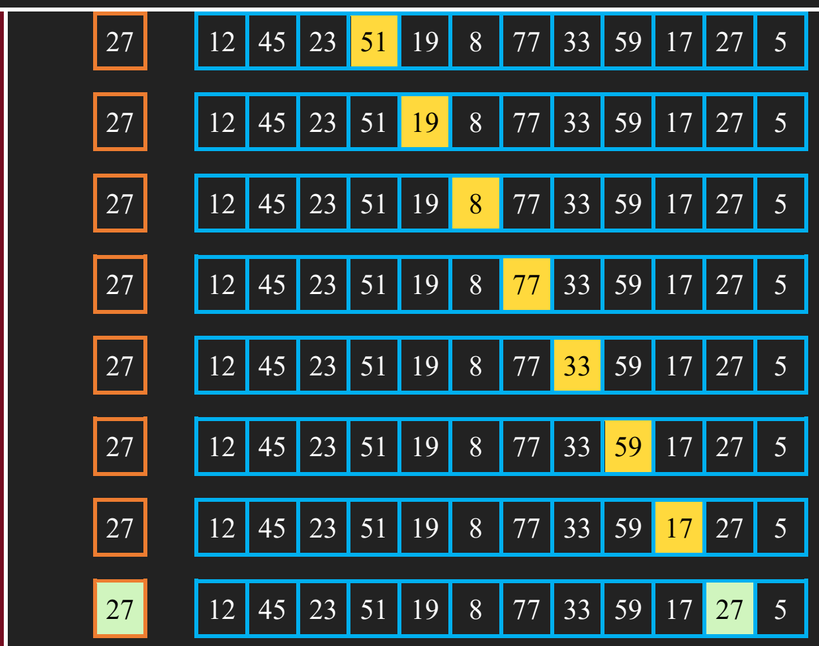
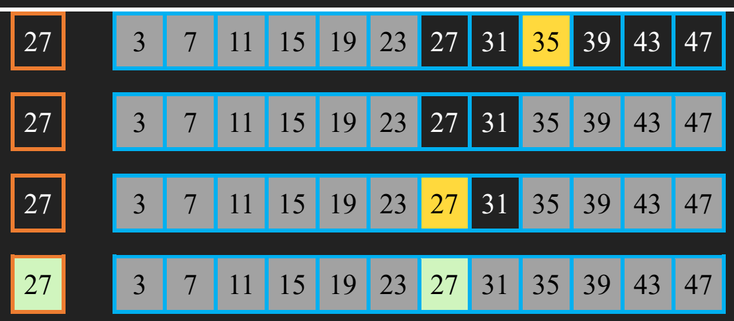
Searching

Properties		Linear Search	Binary Search
Definition		Linear search is a sequential searching algorithm where we start from one end and check every element of the list until the desired element is found.	Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half.
Time Complexity	**Best**	$O(1)$	$O(1)$
	Average	$O(n)$	$O(\log n)$
	Worst	$O(n)$	$O(\log n)$
Space Complexity		$O(1)$	$O(1)$
Applications		• For searching operations in smaller arrays (<100 items).	• In libraries of Java, .Net, C++ STL. • While debugging, the binary search is used to pinpoint the place where the error happens. • Binary search can be used as a building block for more complex algorithms used in machine learning, such as algorithms for training neural networks or finding the optimal hyperparameters for a model. • It can be used for searching in computer graphics such as algorithms for ray tracing or texture mapping. • It can be used for searching a database.
Advantages		• Linear search can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type. • Does not require any additional memory. • It is a well-suited algorithm for small datasets.	• Binary search is faster than linear search, especially for large arrays. • More efficient than other searching algorithms with a similar time complexity, such as interpolation search or exponential search. • Binary search is well-suited for searching large datasets that are stored in external memory, such as on a hard drive or in the cloud.
Disadvantages		• Linear search has a time complexity of $O(n)$, which in turn makes it slow for large datasets. • Not suitable for large arrays.	• The array should be sorted. • Binary search requires that the data structure being searched be stored in contiguous memory locations. • Binary search requires that the elements of the array be comparable, meaning that they must be able to be ordered.
Visualization			





Code Implementation

```c
#include <stdio.h>

int linearSearch(int array[], int size, int data);

int main()
{
  int size, data;
  printf("Enter the size of the array: ");
  scanf("%d", &size);

  int array[size];
  printf("Enter the elements of the array:\n");
  for (int index = 0; index < size; index++)
  {
    scanf("%d", &array[index]);
  }

  printf("Enter the data to search: ");
  scanf("%d", &data);

  int index = linearSearch(array, size, data);
  if (index == -1)
  {
    printf("Element not found\n");
  }
  else
  {
    printf("Element found at index %d\n", index);
  }
```

```c
#include <stdio.h>

int binarySearch(int array[], int size,
int data);

int main()
{
  int size, data;
  printf("Enter the size of the array:
");
  scanf("%d", &size);

  int array[size];
  printf("Enter the elements of the
array:\n");
  for (int index = 0; index < size;
index++)
  {
    scanf("%d", &array[index]);
  }

  printf("Enter the data to search: ");
  scanf("%d", &data);

  int index = binarySearch(array, size,
data);
  if (index == -1)
  {
    printf("Element not found\n");
```

```c
#include <stdio.h>

int binarySearch(int array[], int low, int
high, int data);

int main()
{
  int size, data;
  printf("Enter the size of the array: ");
  scanf("%d", &size);

  int array[size];
  printf("Enter the elements of the
array:\n");
  for (int index = 0; index < size; index++)
  {
    scanf("%d", &array[index]);
  }

  printf("Enter the data to search: ");
  scanf("%d", &data);

  int index = binarySearch(array, 0, size -
1, data);
  if (index == -1)
  {
    printf("Element not found\n");
  }
  else
```

```
  return 0;
}

int linearSearch(int array[], int size, int data)
{
  for (int index = 0; index < size; index++)
  {
    if (array[index] == data)
    {
      return index;
    }
  }
  return -1;
}
```

```
  }
  else
  {
    printf("Element found at index %d\n",
index);
  }
  return 0;
}

int binarySearch(int array[], int size,
int data)
{
  int low = 0, high = size - 1, mid;
  while (low <= high)
  {
    mid = low + (high - low) / 2;
    if (data == array[mid])
    {
      return mid;
    }
    else if (data < array[mid])
    {
      high = mid - 1;
    }
    else
    {
      low = mid + 1;
    }
  }
  return -1;
}
```

```
  {
    printf("Element found at index %d\n",
index);
  }
  return 0;
}

int binarySearch(int array[], int low, int
high, int data)
{
  if (high >= low)
  {
    int mid = low + (high - low) / 2;
    if (array[mid] == data)
    {
      return mid;
    }
    else if (array[mid] > data)
    {
      return binarySearch(array, low, mid -
1, data);
    }
    else
    {
      return binarySearch(array, mid + 1,
high, data);
    }
  }
  return -1;
}
```