CSE3318: Search

Linear Search/Binary Search

by
Dr. Bhanu Jain

All slides are based on: ***Introduction to Algorithms***, by Thomas H. Cormen, Charles E. Leiserson, Ronald E. Rivest, Clifford Stein,3rd edition (CLRS)

Linear Search

- **Definition**:
  - A straightforward method that checks each element in the array one by one.
- **Efficiency**:
  - Time Complexity: **O(n)**(slower for large datasets).
  - Best for **unsorted or small datasets**.
- **Features**:
  - Easy to implement.
  - Does not require the data to be sorted.

Linear Search vs Binary Search

Feature	Linear Search	Binary Search
Time Complexity	O(n)	O(logn)
Input Requirement	Unsorted or sorted	Must be sorted
Speed	Slower for large data	Faster for large data
Implementation	Simple	Slightly complex

Linear Search

LINEAR-SEARCH(A, n, x)

1. for i ← 1 to n do

2. if A[i] == x then

3. return i

4. return NIL

1.Input:

- A: An array of size n.
- n: The number of elements in the array.
- x: The target element to search for.

2.Output:

- Returns the index i of the first occurrence of x in A if found.
- Returns NIL if x is not present in the array.

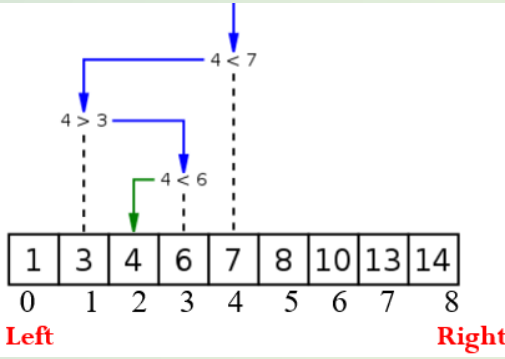
3.Steps:

- Iterate through the array from index 1 to n.
- Compare each element A[i] with the target x.
- If a match is found, return the index i.
- If the loop completes without finding x, return NIL.

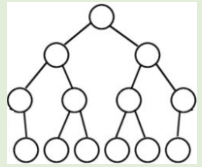
Features:

- **Time Complexity**: O(n) in the worst case (when xxx is at the end or not present).
- **Space Complexity**: O(1), as no extra memory is used.
- **Use Case**: Works for **unsorted** arrays or small datasets where binary search is unnecessary.

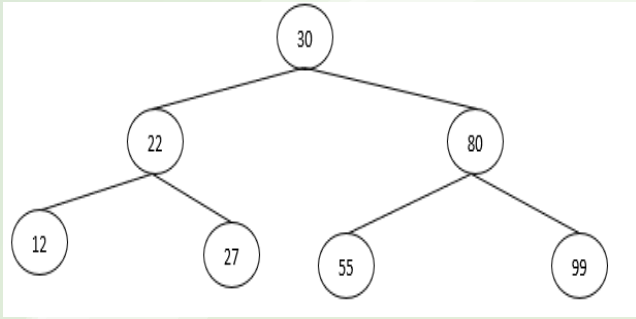
Binary Search, Binary Tree, and Binary Search Tree



- **Binary Search**:
  - Efficient search **algorithm for sorted arrays**, using a divide-and-conquer approach.
  - Time complexity: O(logn).
  - Key feature: Halves the search space with each step.
- **Binary Tree**:
  - **Hierarchical data structure** with nodes, where **each node has at most two children** (left and right).
  - Applications: Expression trees, hierarchical data, and more.



- **Binary Search Tree (BST)**:
  - **A special type of binary tree** where the left child contains smaller values, and the right child contains larger values.
  - Enables fast search, insertion, and deletion operations.



Binary Search Tree

- **Definition**:
  A binary tree where each node follows these rules:
  - Left subtree contains nodes with values smaller than the parent.
  - Right subtree contains nodes with values larger than the parent.
- **Operations**:
  - **Search**: Traverse left or right based on comparison with the root.
  - **Insertion**: Place the new node in the correct position to maintain order.
  - **Deletion**: Three cases to handle:
    - Node with no children: Remove directly.
    - Node with one child: Replace with the child.
    - Node with two children: Replace with the smallest node in the right subtree (inorder successor).
- **Time Complexity**:
  - Best/Average case: O(logn) (balanced tree).
  - Worst case: O(n) (unbalanced tree).
  - .

Binary Search Tree

- **Applications**:
  - Dynamic sets and lookup tables.
  - Database indexing.
  - Sorting algorithms like Tree Sort.
- **Advantages**:
  - Provides ordered storage.
  - Efficient for search and update operations.
- **Drawback**:
  - Can become unbalanced, leading to O(n) performance for skewed trees.
  - Balanced variants like AVL and Red-Black Trees solve this issue.
  - .

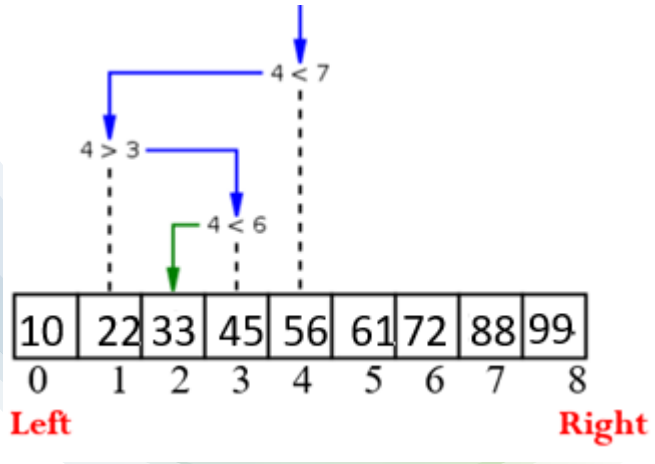
Binary Search

- **Features**:
  - Works only on **sorted arrays or lists**.
  - Eliminates half of the search space at each step by comparing the target with the middle element.
- **Algorithm Steps**:
  - Find the **middle element** of the array.
  - Compare the **target with the middle element**:
    - If equal, target found.
    - If smaller, repeat on the left half.
    - If larger, repeat on the right half.
  - **Repeat until** the element is **found or** the array **size** becomes **zero**.

Binary Search

- **Time Complexity**:
  - Best case: O(1) (target is the middle element).
  - Average/Worst case: O(logn).
- **Applications**:
  - Searching in sorted arrays.
  - Efficient retrieval in sorted datasets.
- **Advantages**:
  - Extremely fast for large datasets.
  - Minimal memory usage since it doesn’t require additional storage.

Binary Search




```
01 #include <stdio.h>
02 #define SIZE 9
03 int main(void)
04 {int intarr[SIZE], target, i, left, right, mid;
05 printf("Please input %d integers in ascending order.\n", SIZE);
06 for(i = 0; i<SIZE; i++)
07 {  scanf("%d", &intarr[i]); }
08 printf("Please input a target (to be searched) value.\n");
09 scanf("%d", &target);
10 left = 0;
11 right = SIZE-1;
12 while(left <= right)
13      {   mid = (left+right)/2;
14      printf ("left= %d, mid= %d, right= %d\n",left,mid,right);
15       if(intarr[mid] == target)      { printf("The index of the target in the array is %d.\n", mid);   break;  }
16       else if (target > intarr[mid]) { left = mid + 1;  }
17       else                           { right = mid - 1; }
18      }
19 if(left > right)   printf("The target is not in the array.\n");
20 return 0;      }
21
```

```
Please input 9 integers in ascending order.
10 22 33 45 56 61 72 88 99
Please input a target (to be searched) value.
72
left= 0, mid= 4, right= 8
left= 5, mid= 6, right= 8
The index of the target in the array is 6.
```

#define SIZE 9

- is a **preprocessor directive** that defines a **macro** named SIZE with the value 9
- used to define **constants** in a program.
- Unlike variables, macros do not use memory
- does a direct replacement, so there's no type associated with SIZE

Binary Search

12	22	34	47	55	62	78	89	91	101	117	125	Elements
0	1	2	3	4	**5**	6	7	8	9	10	11	Index
Left					mid						right	

```
01#include <stdio.h>
02#define SIZE 12
03int main(void)
04{int intarr[SIZE], target, i, left, right, mid;
05printf("Please input %d integers in ascending order.\n", SIZE);
06for(i = 0; i<SIZE; i++)
07{  scanf("%d", &intarr[i]); }
08printf("Please input a target (to be searched) value.\n");
09scanf("%d", &target);
10left = 0;
11right = SIZE-1;
12while(left <= right)
13     {    mid = (left+right)/2;
14     printf ("left= %d, mid= %d, right= %d\n",left,mid,right);
15      if(intarr[mid] == target)      { printf("The index of the target in the array is %d.\n", mid);   break;  }
16      else if (target > intarr[mid]) { left = mid + 1;  }
17      else                           { right = mid - 1; }
18     }
19if(left > right)   printf("The target is not in the array.\n");
20return 0;      }
21
```

```
Please input 12 integers in ascending order.
12 22 34 47 55 62 78 89 91 101 117 125
Please input a target (to be searched) value.
125
left= 0, mid= 5, right= 11
left= 6, mid= 8, right= 11
left= 9, mid= 10, right= 11
left= 11, mid= 11, right= 11
The index of the target in the array is 11.
```

Binary Search

```
01 #include <stdio.h>
02 #define SIZE 9
03 int main(void)
04 {int intarr[SIZE], target, i, left, right, mid;
05 printf("Please input %d integers in ascending order.\n", SIZE);
06 for(i = 0; i<SIZE; i++)
07 {  scanf("%d", &intarr[i]); }
08 printf("Please input a target (to be searched) value.\n");
09 scanf("%d", &target);
10 left = 0;
11 right = SIZE-1;
12 while(left <= right)
13      {   mid = (left+right)/2;
14      printf ("left= %d, mid= %d, right= %d\n",left,mid,right);
15       if(intarr[mid] == target)      { printf("The index of the target in the array is %d.\n", mid);   break;  }
16       else if (target > intarr[mid]) { left = mid + 1;  }
17       else                           { right = mid - 1; }
18      }
19 if(left > right)
20 {    printf ("Left= %d, mid= %d, right= %d\n",left,mid,right);
21     printf("The target is not in the array.\n");
   }
   return 0;      }
```

12	22	34	47	55	62	78	89	91	Elements
0	1	2	3	**4**	**5**	6	7	8	Index
left				**mid**				right	

```
Please input 9 integers in ascending order.
12 22 34 47 55 62 78 89 91
Please input a target (to be searched) value.
92
left= 0, mid= 4, right= 8
left= 5, mid= 6, right= 8
left= 7, mid= 7, right= 8
left= 8, mid= 8, right= 8
Left= 9, mid= 8, right= 8
The target is not in the array.
```

#define SIZE 9

- is a **preprocessor directive** that defines a **macro** named SIZE with the value 9
- used to define **constants** in a program.
- Unlike variables, macros do not use memory
- does a direct replacement, so there's no type associated with SIZE

Binary Search

```
01 #include <stdio.h>
02 #define SIZE 12
03 int main(void)
04 {int intarr[SIZE], target, i, left, right, mid;
05 printf("Please input %d integers in ascending order.\n", SIZE);
06 for(i = 0; i<SIZE; i++)
07 {  scanf("%d", &intarr[i]); }
08 printf("Please input a target (to be searched) value.\n");
09 scanf("%d", &target);
10 left = 0;
11 right = SIZE-1;
12 while(left <= right)
13      {   mid = (left+right)/2;
14      printf ("left= %d, mid= %d, right= %d\n",left,mid,right);
15       if(intarr[mid] == target)      { printf("The index of the target in the array is %d.\n", mid);   break;  }
16       else if (target > intarr[mid]) { left = mid + 1;  }
17       else                           { right = mid - 1; }
18      }
19 if(left > right)
20 {    printf ("Left= %d, mid= %d, right= %d\n",left,mid,right);
21    printf("The target is not in the array.\n");
   }
   return 0;      }
```

12	22	34	47	55	62	78	89	91	101	117	125	Elements
0	1	2	3	4	**5**	6	7	8	9	10	11	Index
left											right	

```
Please input 12 integers in ascending order.
12 22 34 47 55 62 78 89 91 101 117 125
Please input a target (to be searched) value.
10
left= 0, mid= 5, right= 11
left= 0, mid= 2, right= 4
left= 0, mid= 0, right= 1
Left= 0, mid= 0, right= -1
The target is not in the array.
```

#define SIZE 9

- is a **preprocessor directive** that defines a **macro** named SIZE with the value 9
- used to define **constants** in a program.
- Unlike variables, macros do not use memory
- does a direct replacement, so there's no type associated with SIZE

Binary Search

- In binary search, at most $\log_2(n)$ comparisons are made.

- If the sorted array has one million integers (2^{20})
  - Linear search can make a million comparison operations
  - Binary search will make around 20 comparisons

  - We need to keep an eye on the performance of algorithms, a very important aspect of programming.

The END!