

Experiment 1

Student name: *Aditya Kumar (24MAI14003)*

Subject: *Advanced Data Structures and Algorithms Lab (24CSH-622)*

– Professor: *Dr. Manjit Singh*

Date: *02 Aug, 2024*

Aim: Write a program for the implementation of the following searching techniques on Linear Data Structures

1. Linear Search
2. Binary Search

1 Theory

1.1 Linear Search

Linear search is a simple search algorithm that checks every element in a list or array sequentially until the target element is found or the end of the list is reached. It has a time complexity of $O(n)$ in the worst case, where n is the number of elements in the list. Linear search is useful when dealing with unsorted lists or arrays.

1.2 Binary Search

Binary search is a more efficient search algorithm that works on sorted lists or arrays. It repeatedly divides the search interval in half until the target element is found or the search interval is empty. The time complexity of binary search is $O(\log n)$ in the worst case. Binary search is useful when dealing with sorted lists or arrays.

2 Algorithm

2.1 Linear Search

1. Start at the beginning of the list.
2. Compare the target element with the current element.
3. If the target element is found, return the index of the element.
4. If the target element is not found, move to the next element.
5. Repeat steps 2-4 until the end of the list is reached.
6. If the target element is not found, return -1.

2.2 Binary Search

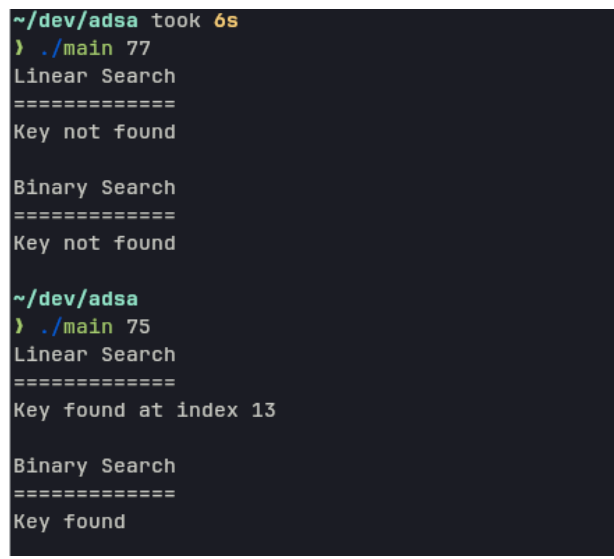
1. Initialize the left and right pointers to the start and end of the list, respectively.
2. Calculate the middle index as the average of the left and right pointers (rounded down).
3. Compare the target element with the middle element.
4. If the target element is found, return the middle index.
5. If the target element is smaller than the middle element, update the right pointer to the middle index - 1.
6. If the target element is larger than the middle element, update the left pointer to the middle index + 1.
7. Repeat steps 2-6 until the target element is found or the left pointer is greater than the right pointer.
8. If the target element is not found, return -1.

3 Code

```
1  import <algorithm>;
2  import <optional>;
3  import <print>;
4  import <ranges>;
5  import <vector>;
6
7  auto b_search(const std::vector<ssize_t> &vec, const ssize_t &key) -> bool {
8      auto temp{vec};
9      std::ranges::sort(temp);
10     auto mid{temp.size() / 2};
11     auto beg{0ul}, end{temp.size()};
12     while (beg <= end) {
13         mid = (beg + end) / 2;
14         if (temp.at(mid) == key)
15             return true;
16         else if (key < temp.at(mid)) {
17             end = mid - 1;
18         } else {
19             beg = mid + 1;
20         }
21     }
22     return false;
23 }
24
25 auto l_search(const std::vector<ssize_t> &vec, const ssize_t &key)
26     -> std::optional<size_t> {
27     auto index{0u};
28     for (const auto &e : vec) {
29         if (e == key) {
30             return index;
31         }
32         ++index;
```

```
33     }
34     return std::nullopt;
35 }
36
37 int main(int argc, char **argv) {
38     std::vector<ssize_t> vec{5, 3, 5, 4, 6, 5, 7, 45, 6,
39                             45, 6, 5, 6756, 75, 54, 634, 5, 3};
40     auto key{std::stoi(argv[1])};
41     // ssize_t key{77};
42     std::println("Linear Search");
43     std::println("=====");
44     auto result{l_search(vec, key)};
45     if (result.has_value()) {
46         std::println("Key found at index {}", result.value());
47     } else
48         std::println("Key not found");
49     auto b_result{b_search(vec, key)};
50     std::println("\nBinary Search");
51     std::println("=====");
52     b_result ? std::println("Key found") : std::println("Key not found");
53     return 0;
54 }
```

4 Output



```
~/dev/adsa took 6s
} ./main 77
Linear Search
=====
Key not found

Binary Search
=====
Key not found

~/dev/adsa
} ./main 75
Linear Search
=====
Key found at index 13

Binary Search
=====
Key found
```

5 Learning Outcomes

1. Understand the concepts and differences between linear search and binary search.
2. Implement both algorithms in their preferred programming language.
3. Apply linear search and binary search to solve problems involving searching for elements in lists or arrays.
4. Recognize the importance of using binary search for sorted lists or arrays due to its more efficient time complexity compared to linear search.