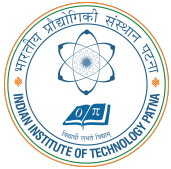
CS1101: Foundations of Programming

Searching



Dept. of Computer Science & Engineering

Indian Institute of Technology Patna

Searching

- Check if a given element, usually called key, is present in a list / array
- Example: Searching for a student record, roll number can be the key
- Two broad approaches will be looked into:
  - List / array is not-sorted: We have no other option but to check every element present in the list. (Linear search)
  - List / array is sorted: Though the previous method works here, it will be inefficient. We can use binary search

Linear search

- **Basic idea**
  - Start from the 1st element of the array
  - Inspect each element in sequence and check if it matches with the key
  - If a match is found, then report the element is present
  - If no match is found, report that the key does not exist. We can return special value such as -1 to indicate such situation

Linear search

- **Returns the index of the array if key is present, else -1**

```
int linearSearch(int a[], int size, int key){
   int loc = 0;
   while ((loc < size) && (key != a[loc]))
     loc++;
   if(loc < size)
     return loc;
   return -1
}
```

Time complexity of linear search

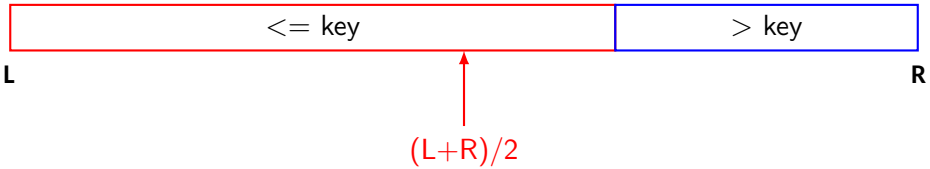
- This is measured using the number of basic operations that needs to be performed before termination
- Example: Number of comparisons that we perform here
- Assume that there are n element in the array
  - Best case: match found in the first comparison. (1 comparison)
  - Worst case: no match found or the last element matches (n comparisons)
  - Average case: $\frac{(n+1)}{2}$ comparisons

Binary search

- Binary search is applicable if the array is sorted
- Basic idea
  - Look for the key at the mid position of the array
  - If it matches, then we are done. If key is less than mid element, then we need to search on the left of array. Otherwise, right half of the array will be looked into
- In every step, we reduce the number of elements by a factor of 2

Basic strategy

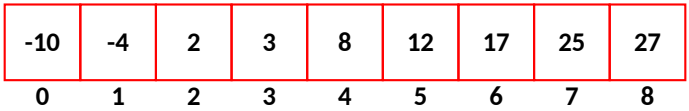
- Initially the search window is the whole array, that is, between L & R
- We compare the key with the middle element and decide which side to search



Binary search

```
int bsearch(int x[], int size, int key){
   int l = 0, r = size - 1, mid;
   while (l != r){
     mid = (l+r)/2;
     if(key <= x[mid]) r = mid;
     else l = mid+1;
   }
   if(key == x[l]) return l;
   else return -1;
}
```

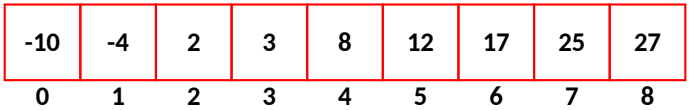
Binary search-1



bsearch(x, 9, 2)

L	R	mid	condition

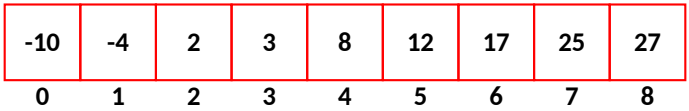
Binary search-1



bsearch(x, 9, 2)

L	R	mid	condition
0	8	4	2 <= 8

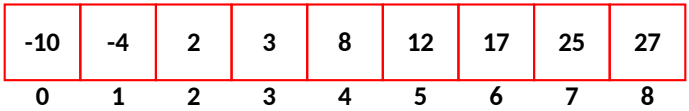
Binary search-1



bsearch(x, 9, 2)

L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2

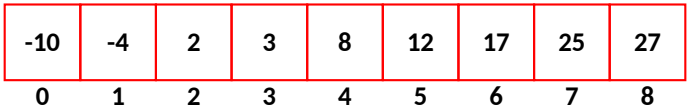
Binary search-1



bsearch(x, 9, 2)

L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4

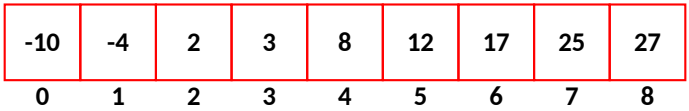
Binary search-1



bsearch(x, 9, 2)

L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4
2	2		Terminated

Binary search-1

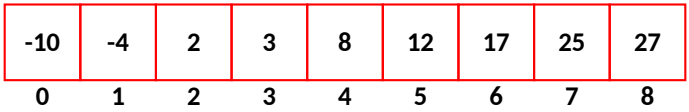


bsearch(x, 9, 2)

L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4
2	2		Terminated

x[2] == 2 ? TRUE

Binary search-2



bsearch(x, 9, 2)

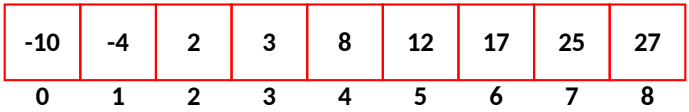
L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4
2	2		Terminated

x[2] == 2 ? TRUE

bsearch(x, 9, 1)

L	R	mid	condition

Binary search-2



bsearch(x, 9, 2)

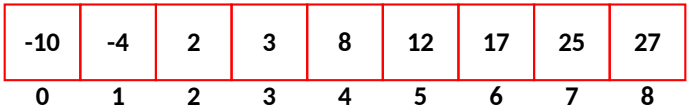
L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4
2	2		Terminated

x[2] == 2 ? TRUE

bsearch(x, 9, 1)

L	R	mid	condition
0	8	4	1 <= 8

Binary search-2



bsearch(x, 9, 2)

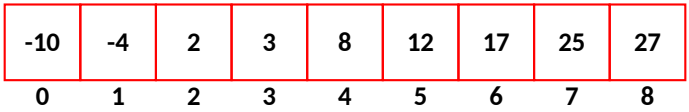
L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4
2	2		Terminated

x[2] == 2 ? TRUE

bsearch(x, 9, 1)

L	R	mid	condition
0	8	4	1 <= 8
0	4	2	1 <= 2

Binary search-2



bsearch(x, 9, 2)

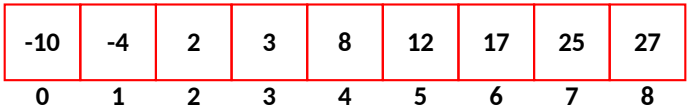
L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4
2	2		Terminated

x[2] == 2 ? TRUE

bsearch(x, 9, 1)

L	R	mid	condition
0	8	4	1 <= 8
0	4	2	1 <= 2
0	2	1	1 <= -4

Binary search-2



bsearch(x, 9, 2)

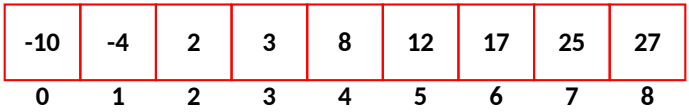
L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4
2	2		Terminated

x[2] == 2 ? TRUE

bsearch(x, 9, 1)

L	R	mid	condition
0	8	4	1 <= 8
0	4	2	1 <= 2
0	2	1	1 <= -4
2	2		Terminated

Binary search-2



bsearch(x, 9, 2)

L	R	mid	condition
0	8	4	2 <= 8
0	4	2	2 <= 2
0	2	1	2 <= -4
2	2		Terminated

x[2] == 2 ? TRUE

bsearch(x, 9, 1)

L	R	mid	condition
0	8	4	1 <= 8
0	4	2	1 <= 2
0	2	1	1 <= -4
2	2		Terminated

x[2] == 1 ? FALSE

Binary search: alternative version

```
int bsearch(int x[], int size, int key){
   int l = 0, r = size - 1, mid;
   while (l <= r){
     mid = (l+r)/2;
     if(key == x[mid]) return mid;
     if(key < x[mid]) r = mid - 1;
     else l = mid+1;
   }
   return -1;
}
```

Binary search: recursive version

- The algorithm is called recursively by adjusting the left or right pointers as applicable
- The base condition is: the element is found, or the left and right pointers cross

```
int bsearch(int x[], int l, int r, int key){
   int mid;
   if(l <= r){
     mid = (l+r)/2;
     if(key == x[mid]) return mid;
     if(key < x[mid]) return bsearch(x, l, mid-1, key);
     else return bsearch(x, mid+1, r, key);
   }
   return -1;
}
```

Time complexity

- Consider an array of size 1000 sorted in increasing order
- Linear search: need to check every element, in the worst case we need to perform 1000 comparisons
- Binary search: first we compare with 500th element and discard 500 elements, next we discard 250 elements, and so on. Roughly, in 10 steps you end up with a single element $(2^{10} = 1024)$. Approximately 10 comparisons are need in the worst case
- If size of the array in n, linear search will need n comparisons, and binary search will require $\lceil \log_2 n \rceil$ comparisons