UTM
UNIVERSITI TEKNOLOGI MALAYSIA

DATA STRUCTURE & ALGORITHM

SEARCHING TECHNIQUES 2

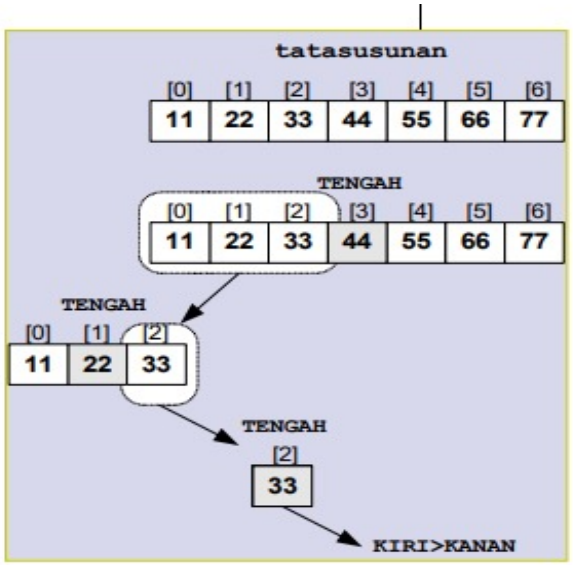
Nor Bahiah Hj Ahmad & Dayang Norhayati A.Jawawi
School of Computing

Binary Search

- The drawbacks of sequential search algorithm is having to traverse the entire list, **O(n)**
- Sorting the list minimize the cost of traversing the whole data set, but we can improve the searching efficiency by using the Binary Search algorithm
- Consider a list in ascending sorted order. For a sequential search, searching is from the beginning until an item is found or the end is reached. Binary search improve the algorithm by removing as much of the data set as possible so that the item is found more quickly.
- Search process is started at the middle of the list, then the algorithm determine which half the item is in (because the list is sorted).
  - It divides the working range in half with a single test. By repeating the procedure, the result is an efficient search algorithm, **($\log_2$ n).**

Binary Search

- The algorithm starts by comparing the search key with the element at the middle of the list
  - If the value matches, it will be return to the calling function with (index = MIDDLE)
  - If the search key is smaller than the middle element, search will be focused on the elements between the first element to the element before the middle element (MIDDLE -1)
  - If the search key is not found, the element at the middle of the first element and the MIDDLE –1 element will be compared with the search key
  - If the value is bigger, search will only be focused on the elements between the second MIDDLE element to the first MIDDLE element.
- Search is repeated until the searched key is found or the last element in the subset is traversed.

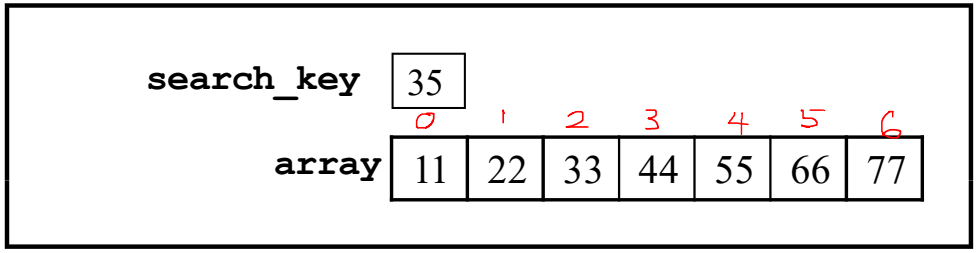


UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Binary Search Function

```
int binary_search( int  search_key, int array_size,
                       const int array [] )
{   bool found = false;
    int index = -1 //-1 means record not found
    int MIDDLE,
    LEFT = 0,
    RIGHT = array_size-1;
    while (( LEFT<= RIGHT ) && (!found))
    {    MIDDLE = (LEFT + RIGHT )/ 2;    // Get middle index
         if ( array[MIDDLE] == search_key)
         {     index = MIDDLE;
               found = true;
         }
        else if (array[MIDDLE] > search_key)
              RIGHT= MIDDLE- 1;    // search is focused
                                   // on the left side of list
        else
              LEFT= MIDDLE+ 1      // search is focused
                                 // on the right side of the list
       } //end while
     return index;
}//end function
```

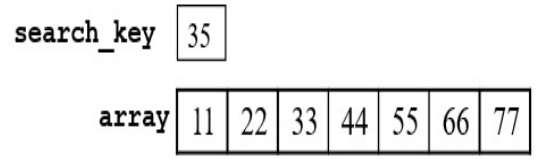
Binary Search on a Sorted List



- Search starts by obtaining the **MIDDLE**index of the array
- assume:
  `search_key` = 35
- **`MIDDLE`**= (0 + 6) / 2
  = 3 { First **`MIDDLE`** index}

Binary Search on a Sorted List..cont

```
else if (array[MIDDLE] > search_key)
    RIGHT= MIDDLE- 1;   // search is focused
                        // on the left side of list
else
    LEFT= MIDDLE+ 1     // search is focused
                        // on the right side of the list
```



y 35 is compared with the element at the fourth
e array, which is `array[3]` with the value **44**.

_key < MIDDLE value, therefore search will be
d on the elements between the first index and the
dex only (index 1 to **MIDDLE-1**)

- rocess to obtain **MIDDLE** index is repeated

  MIDDLE = (0 + 2) / 2

  = 1 { second **MIDDLE** index}

- **Search_key 35** is compared with the element at the second index, which is `array[1]` with the value **22**
  - **search_key > MIDDLE** value, therefore search will be focused only on the elements between the second **MIDDLE** index to the first **MIDDLE** index.

    MIDDLE = (2 + 2) / 2

    = 2 {**MIDDLE** index}

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

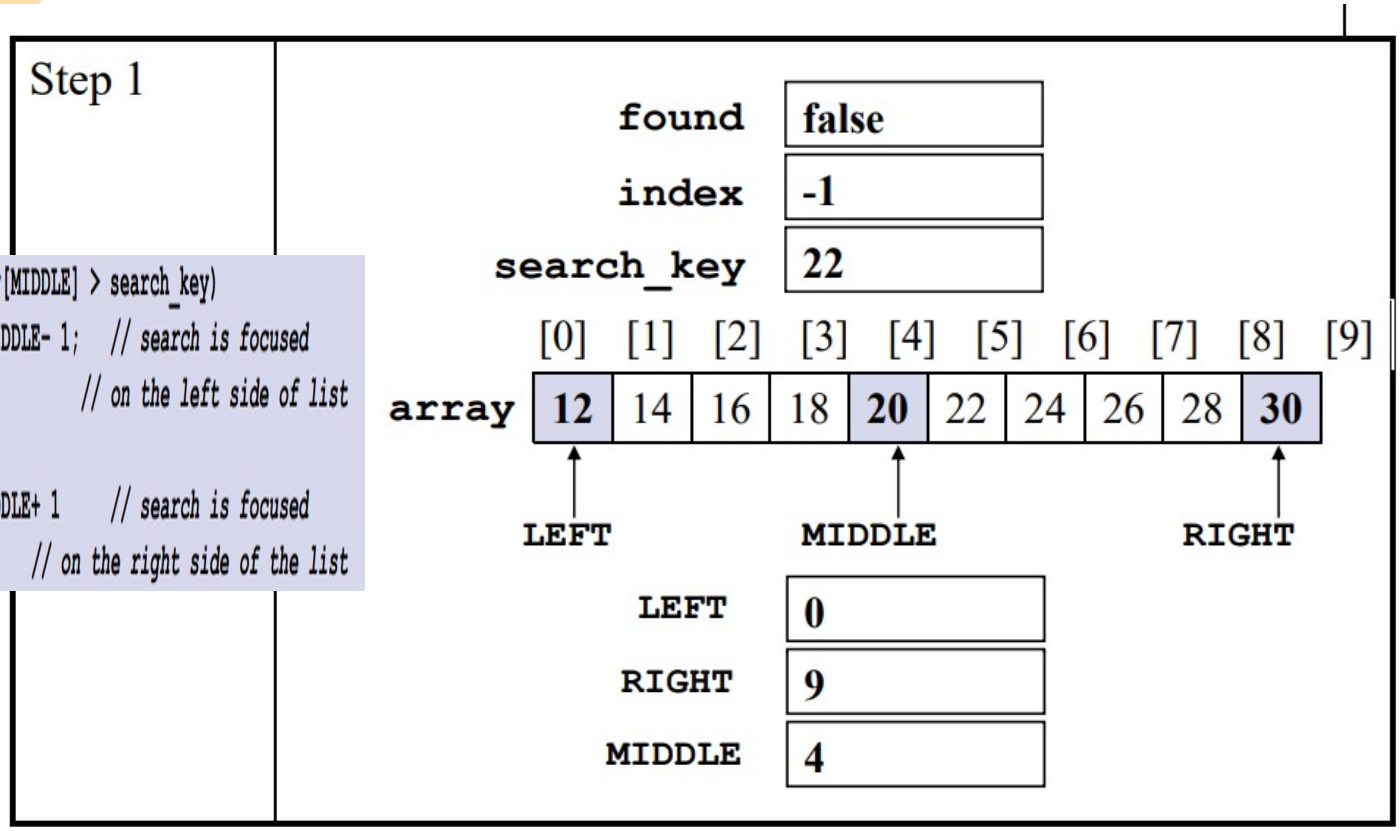
cont...

- Element at the third index, array[2] with the value 33 is not equal to the value of the search key.
- Search process has reached the last element of the traversed subset, therefore search is terminated and assumed fail.
- To search from the list sorted in descending, change operator " **>** " to operator " **<** " to the following statement :

```
else if (array [ MIDDLE ] > search_key)
        RIGHT = MIDDLE - 1;
```

Steps to Execute Binary Search

```
else if (array[MIDDLE] > search_key)
    RIGHT= MIDDLE- 1;   // search is focused
                        // on the left side of list
else
    LEFT= MIDDLE+ 1     // search is focused
                        // on the right side of the list
```

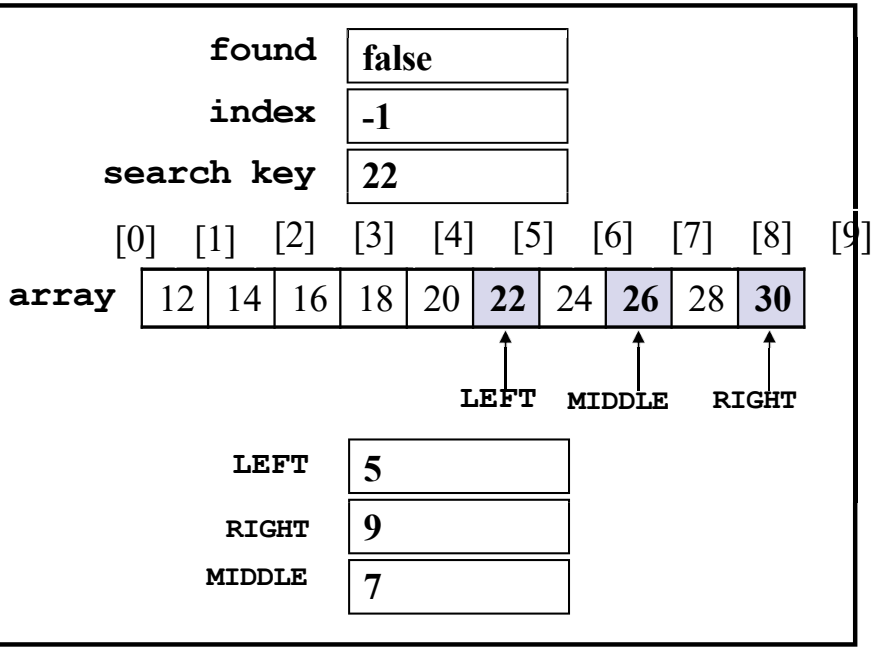


Assume: `search_key` = 22, `array_size` = 10

cont...

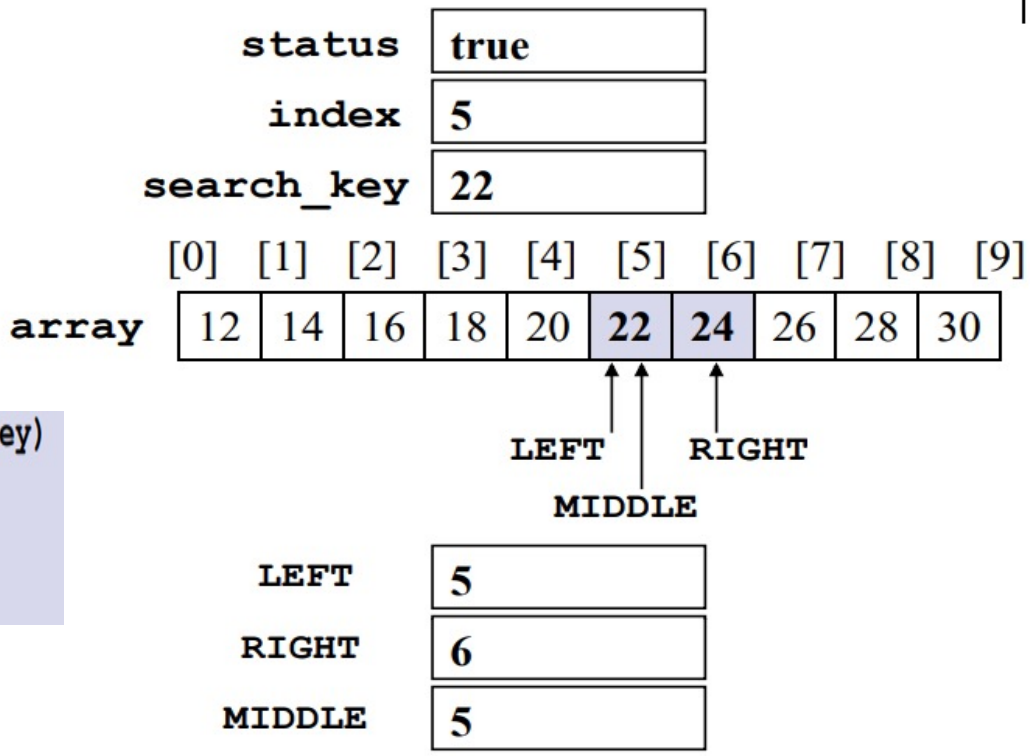
Step 2

```
else if (array[MIDDLE] > search_key)
    RIGHT= MIDDLE- 1;   // search is focused
                        // on the left side of list
else
    LEFT= MIDDLE+ 1     // search is focused
                        // on the right side of the list
```



cont…

Step 3




```
if ( array[MIDDLE] == search_key)
{      index = MIDDLE;
       found = true;
}
```

Binary Search Analysis

- Binary Search starts searching by comparing element in the middle. Thus the searching process start at n/2 for a list size = n.
- If the middle value does not matches with the search key, then the searching area will be reduced to the left or right sublist only. This will reduce the searching area to ½ *n*. From half of the list, the second middle value will be identified. Again, if the middle value does not matches with the search key, the searching area will be reduced to the left or right sub list only. The searching area will reduce ½ (½ *n*).
- The process of looking for middle point and reducing the searching area to the left or right sublist will be repeated until the middle value is equal to the middle value (search key is found) or the last value in sublist has been traverse.
- If the repetition occur *k* times, then at iteration *k* , the searching area is reduced to $(½)^k n$.

Binary Search Analysis

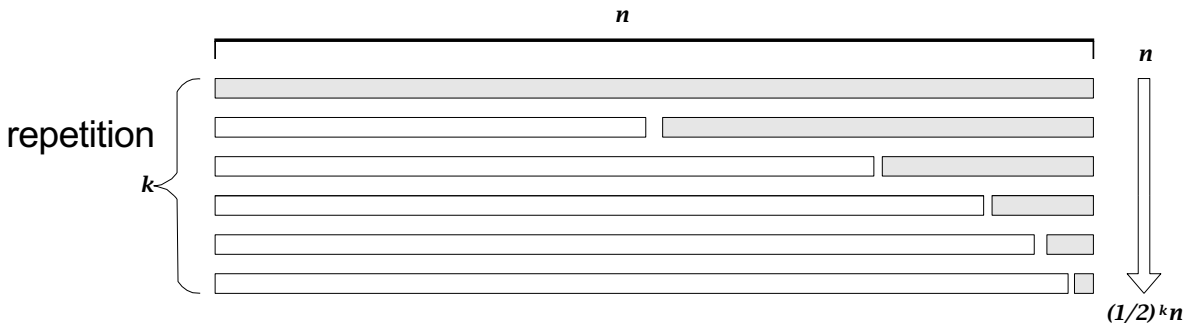


Figure above shows the reducing size for binary search area.

At iteration k for array size = n , searching area will be reduced from n to $(½)^k n$.

Binary Search Analysis

- Best case for binary search happen if the search key is found in the middle array. O(1).
- Worse case for binary search happen if the search key is not in the list or the searching size equal to 1.
- The searching time for worse case is $O(log_2 n)$.

Conclusion

- Searching is a process to locate an element in a list and return the index of the searched element.
- Basic searching techniques : sequential search and binary search.
- Sequential search can be implemented on sorted and unsorted list, while binary search can be implemented only on sorted list.
- Sequential search on sorted data is more efficient than sequential search on unsorted data.
- Binary search is more efficient than sequential search.
- Basic searching techniques explained in this class are only suitable for small sets of data. Hashing and indexing are suitable for searching large sets of data.