Searching

(sequential & binary search)

Overview

- Konsep & Istilah
- Sequential Search
- Binary Search
- Perbandingan Unjuk Kerja *(Performance)*
- Review

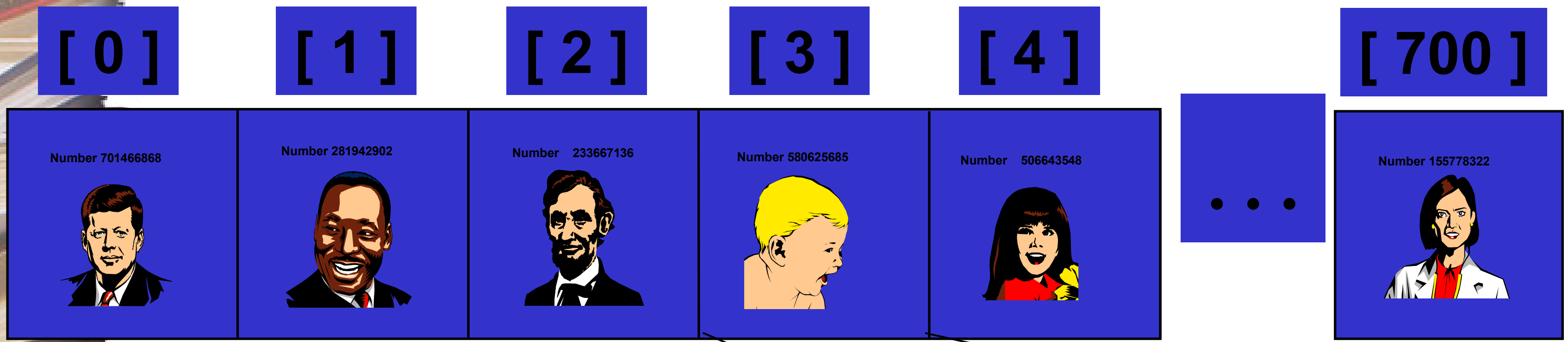
Konsep dan Istilah

- Internal Search – algoritma pencarian yang dilakukan dalam main memory komputer
- External Search –algoritma pencarian yang melibatkan external media menambah main memory

Konsep dan Istilah

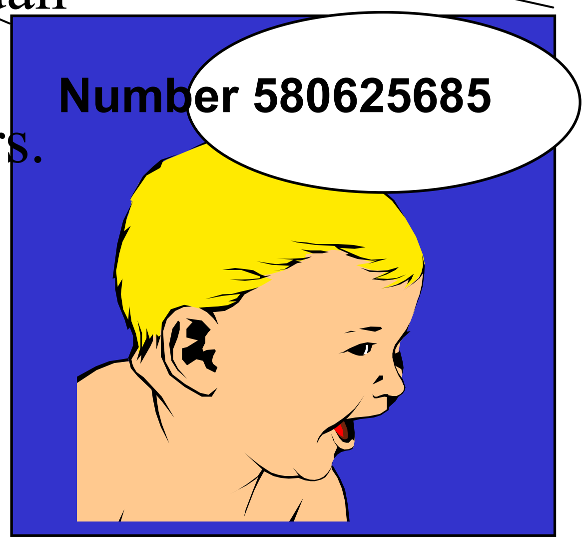
- Key – sebuah subset dari isi sebuah data yang digunakan untuk perbandingan selama proses pencarian
- Big-O Notation – notasi yang digunakan untuk mengindikasikan kenaikan *(Order of growth)* unjuk kerja dari sebuah algoritma searching

Search



Setiap record dalam list memiliki sebuah associated key.
Pada contoh ini, key-nya = ID numbers.

Berikan sebuah key, bagaimana cara menemukan recordnya dari list secara efektif ?



Sequential Search

- Begin at the beginning (or the end)
- Cek seluruh record dalam array atau list, baca satu persatu.
- Temukan record sesuai dengan key yang dicari
- Proses Searching berhenti karena salah satu alasan
  - Success – Found the target key
  - End of List – No more records to compare
- Diaplikasikan pada Array (sorted & unsorted) atau Linked List

Sequential Search - Unsorted

The List

Gordon
Andrew
Michael
Bill
Scott
Luke
Adrian
Dennis

Target Key

Scott

Compare Target Key with
List[0] Gordon
List[1] Andrew
List[2] Michael
List[3] Bill
List[4] Scott – SUCCESS!

5 comparisons required

Sequential Search - Sorted

The List
Adrian
Andrew
Bill
Dennis
Gordon
Luke
Michael
Scott

Target Key

Jeff

Compare Target Key with
List[0] Adrian
List[1] Andrew
List[2] Bill
List[3] Dennis
List[4] Gordon
List[5] Luke – Not Found!

6 comparisons required

Sequential Search Analysis

- Bagaimana worst case dan average case untuk metode sequential search?
- Kita harus mendefinisikan O-notation untuk nilai dari operasi yang dibutuhkan dalam pencarian.
- Jumlah operasi pencarian bergantung pada nilai n, yaitu jumlah elemen dalam list

Worst Case Time for Sequential Search

- Untuk sebuah array dengan *n* elemen, maka worst case time untuk sequential search → requires *n* array accesses: O(*n*).
- Kondisi yang mengharuskan pengecekan terhadap semua elemen array (*n* record) adalah:
  - Record yang dicari berada pada posisi terakhir dari array
  - Setelah pengecekan seluruh elemen array, ternyata record yang dicari tidak berhasil ditemukan dalam array tersebut

Algoritma Sequential Search

1. i ← 0
2. ketemu ← false
3. Selama (tidak ketemu) dan (i < N) kerjakan baris 4
4. Jika (Data[i] = key) maka

        ketemu ← true

    jika tidak

        i ← i + 1

5. Jika (ketemu) maka

        i adalah indeks dari data yang dicari

    jika tidak

        data tidak ditemukan

Binary Search

- Define working range as entire list
- Repeat till done
  - Select the middle record
  - Compare the target key value with the key of the selected “record”
  - Comparison results:
    - Key < middle record : Range = First half
    - Key > middle record : Range = Last half
    - Key = middle record: Success, Done
- Applies only to Sorted Array List

Binary Search

The List

Adrian
Andrew
Bill
Gordon
Luke
Michael
Scott

Target Key

Bill

Compare Target Key with
List[3] Gordon (high)
List[1] Andrew (low)
List[2] Bill (match)

3 comparisons required

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

↑

Find approximate midpoint

Binary Search

Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Is 7 = midpoint key? NO.

Binary Search

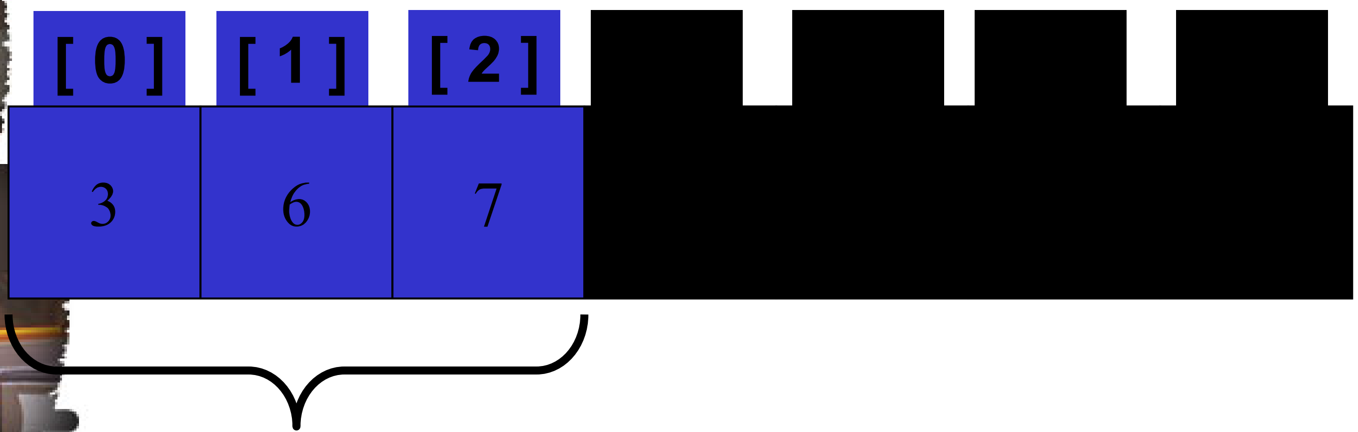
Example: sorted array of integer keys. Target=7.

[0]	[1]	[2]	[3]	[4]	[5]	[6]
3	6	7	11	32	33	53

Is 7 < midpoint key? YES.

Binary Search

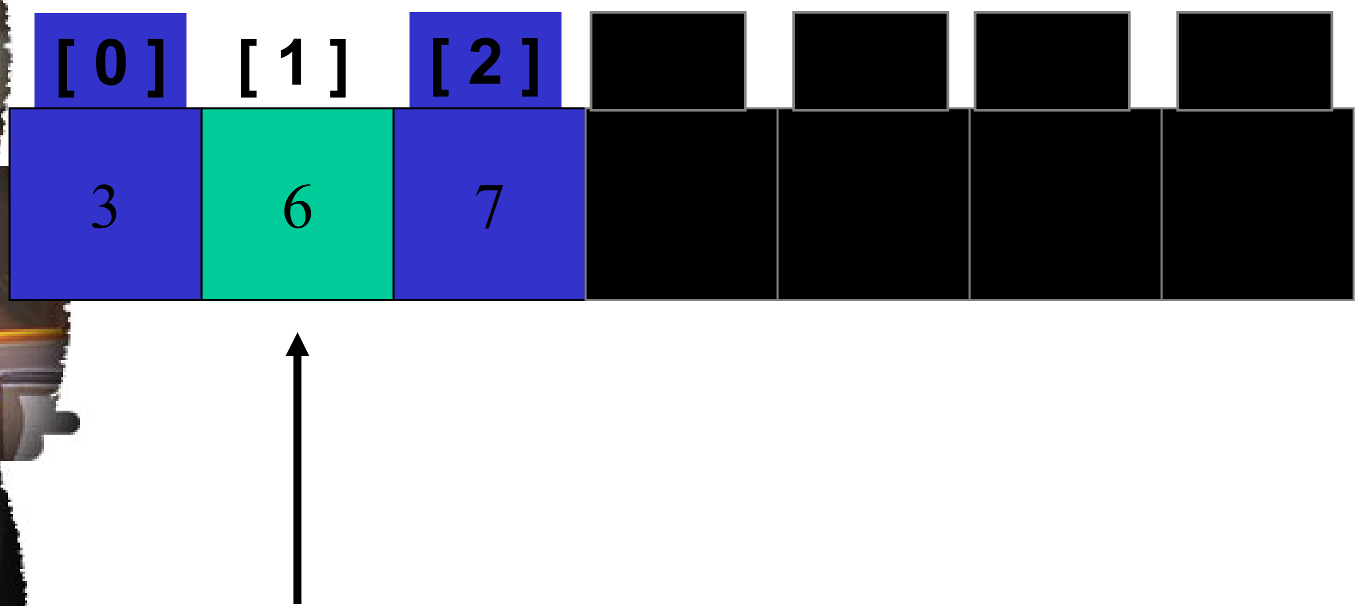
Example: sorted array of integer keys. Target=7.



Search for the target in the area before midpoint.

Binary Search

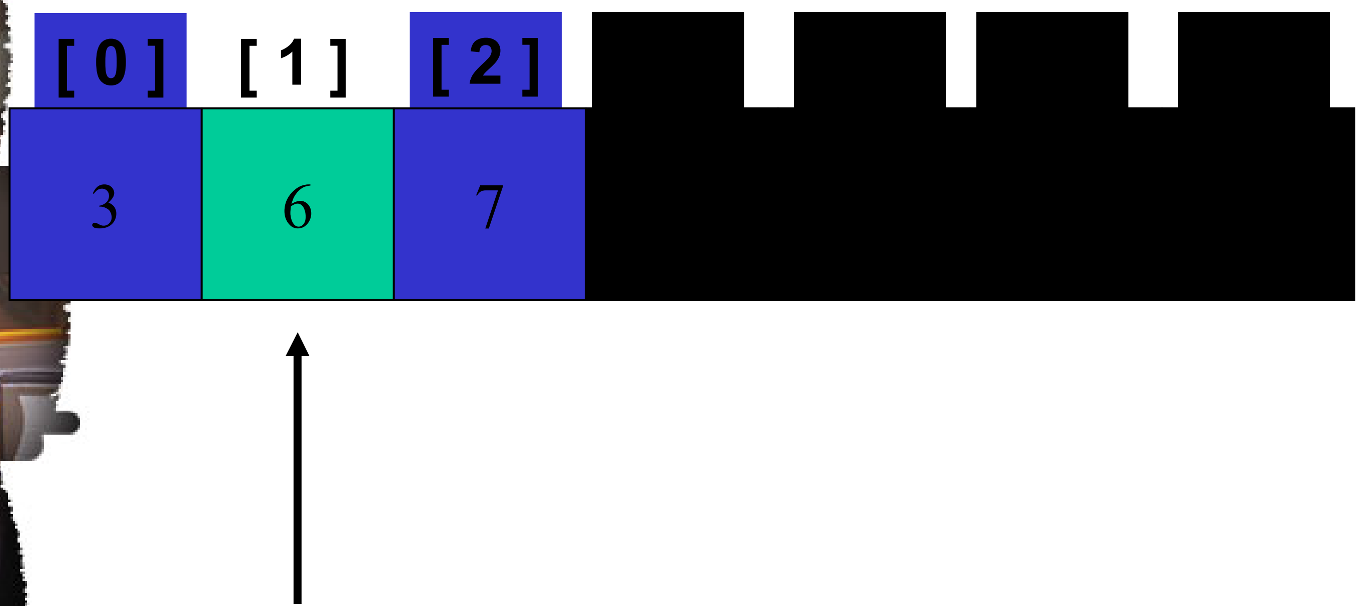
Example: sorted array of integer keys. Target=7.



Find approximate midpoint

Binary Search

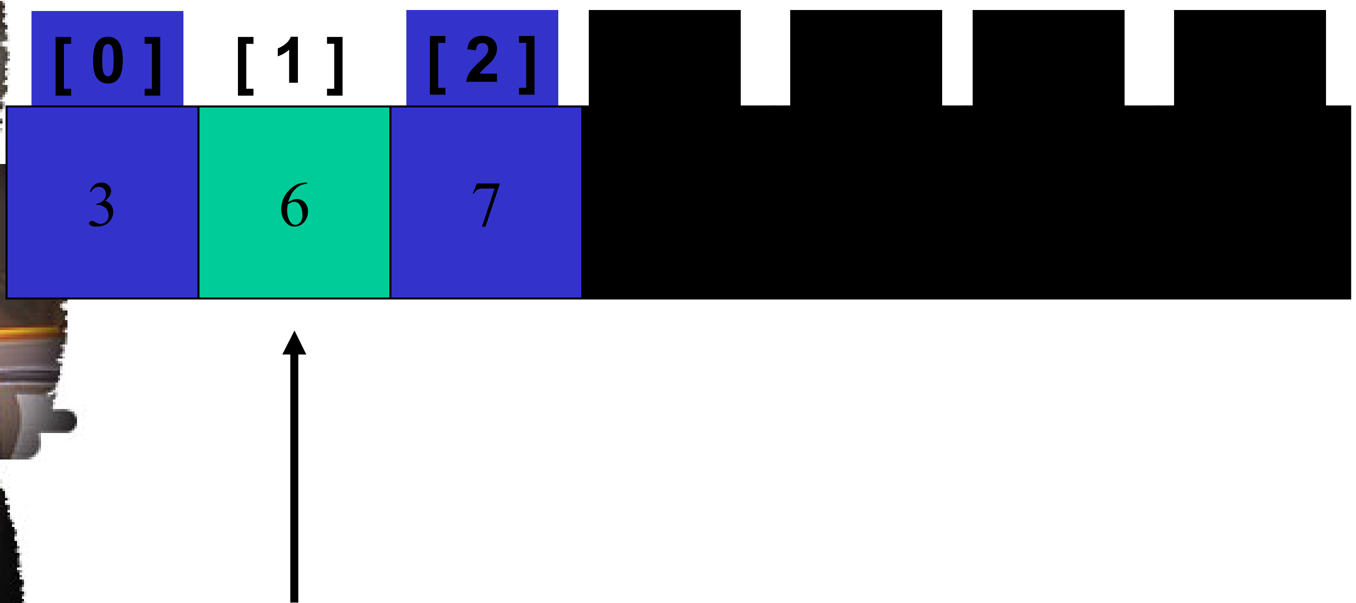
Example: sorted array of integer keys. Target=7.



Target = key of midpoint? NO.

Binary Search

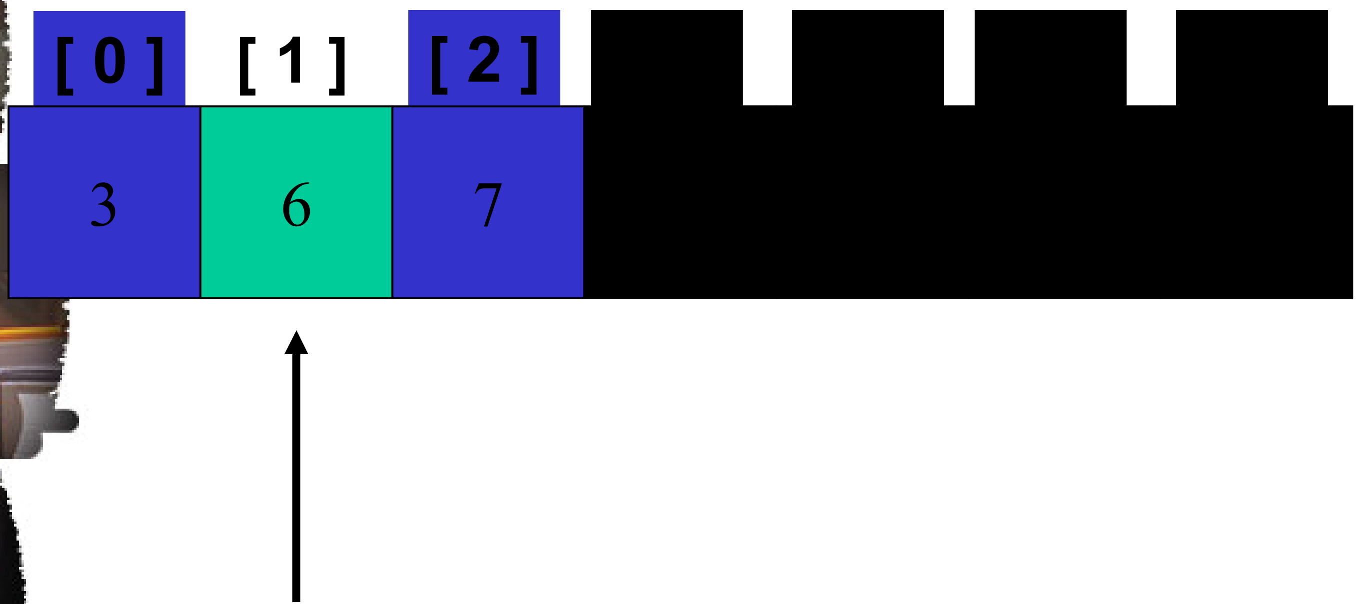
Example: sorted array of integer keys. Target=7.



Target < key of midpoint? NO.

Binary Search

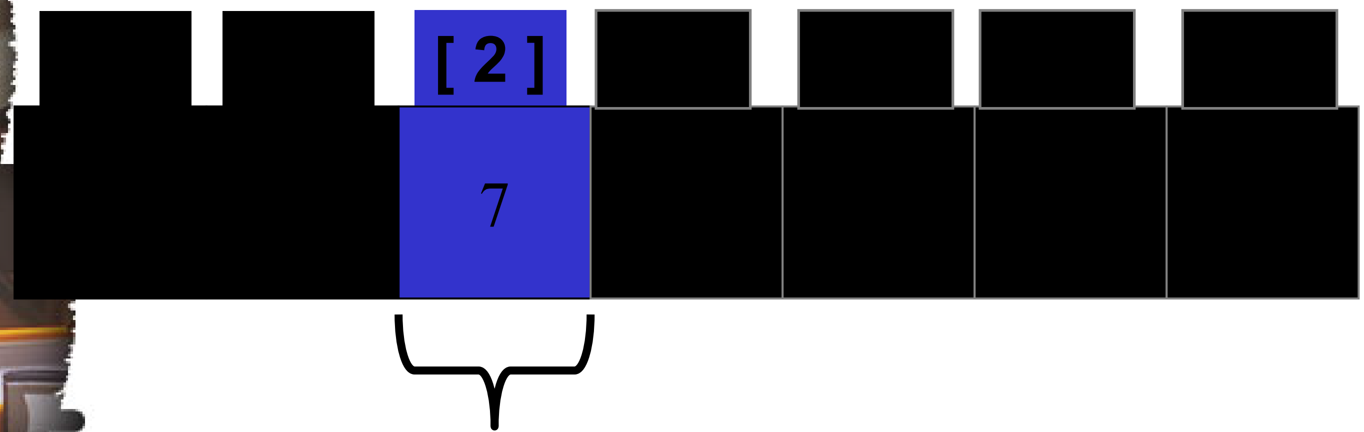
Example: sorted array of integer keys. Target=7.



Target > key of midpoint? YES.

Binary Search

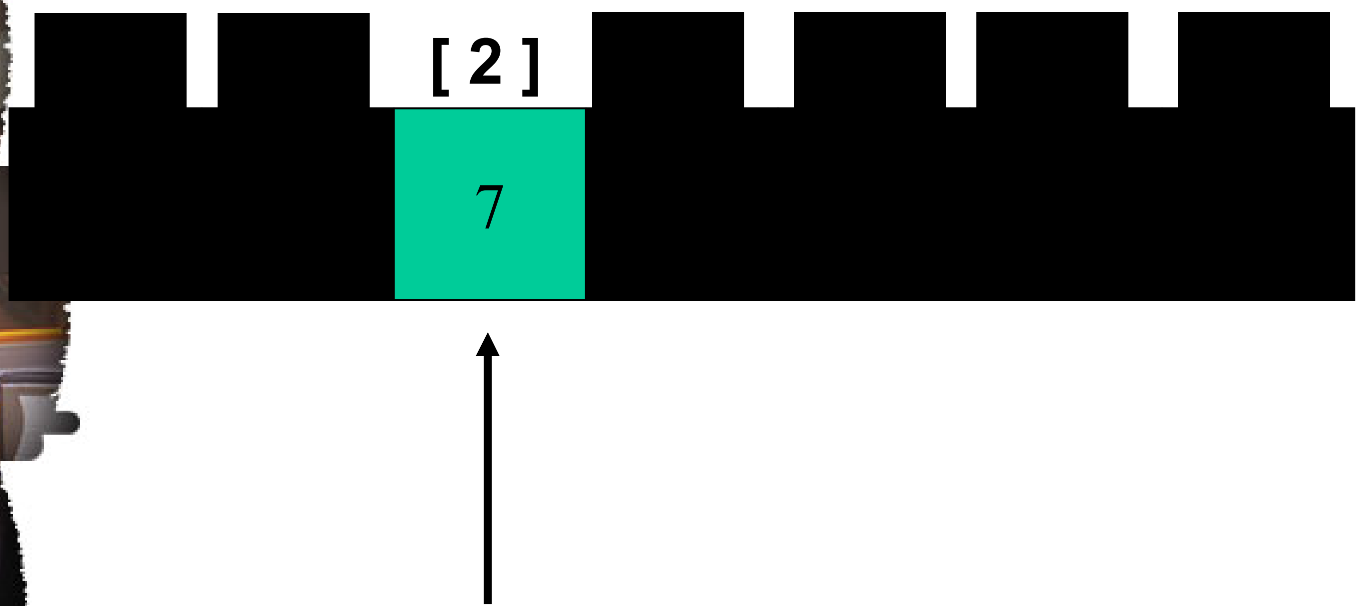
Example: sorted array of integer keys. Target=7.



Search for the target in the area after midpoint.

Binary Search

Example: sorted array of integer keys. Target=7.



Find approximate midpoint.
Is target = midpoint key? YES

Algoritma Binary Search

1. L ← 0
2. R ← N – 1
3. ketemu ← false
4. Selama (L <= R) dan (tidak ketemu) kerjakan baris 5 sampai dengan 8
5. m ← (L + R) / 2
6. Jika (Data[m] = key) maka ketemu ← true
7. Jika (key < Data[m]) maka R ← m – 1
8. Jika (key > Data[m]) maka L ← m + 1
9. Jika (ketemu)

   maka m adalah indeks dari data yang dicari

   jika tidak

   data tidak ditemukan

Binary Search Implementation

```
void search(const int a[ ], size_t first, size_t size, int target, bool& found, size_t&
    location)
{
    size_t middle;
    if(size == 0) found = false;
    else {
            middle = first + size/2;
            if(target == a[middle]){
                    location = middle;
                    found = true;
            }
            else if (target < a[middle])
                  // target is less than middle, so search subarray before middle
                   search(a, first, size/2, target, found, location);
            else
                 // target is greater than middle, so search subarray after middle
                   search(a, middle+1, (size-1)/2, target, found, location);
    }
}
```

Time complexity

- Let T(n) denote the time complexity of binary search algorithm on n numbers.

$$T(n) = \begin{cases} T(\lfloor n/2 \rfloor) + 1 & \text{otherwise} \\ 0 & \text{if } n=1 \end{cases}$$

- We call this formula a **<u>recurrence</u>**.

$$A(n) = \begin{cases} 0 & \text{if } n = 1 \\ A(\lfloor n/2 \rfloor)+1 & \text{if } n > 1 \end{cases}$$

$A(n) = A(2k)$

$= A(2k-1) + 1$

$= A(2k-2) + 1 + 1 = A(2k-2) + 2$

$A(2k-i) + i$

$= A(2k-k) + k$

$= A(20) + k = A(1) + k = 0 + k = k$

$n = 2k \rightarrow k = {}^{2}\log n$

$A(n) = {}^{2}\log n$

Performance Comparisons

Search Algorithm	List Size: 100 elements	List Size: 10,000 elements	List Size: 1,000,000 elements
Sequential Search (Average)	**50**	**5,000**	**500,000**
Binary Search (Maximum)	**$^{2}\log n = 7$** **$2^7 = 128$**	**14** **$2^{14} = 16,384$**	**20** **$2^{20} = 1,048,576$**