

CSE1303 Part A
Data Structures and Algorithms
Summer Semester 2003

Lecture A10 – Elementary Algorithms (Revision)

Kymerly Fergusson

Overview

- Selection Sort
- Insertion Sort
- Linear Search.
- Binary Search.
- Growth Rates.
- Implementation.

2

Selection Sort

First find the largest element in the array and exchange it with the element in the last position, then find the second largest element in array and exchange it with the element in the second last position, and continue in this way until the entire array is sorted.

3

Selection Sort

1	5	0	2	6	3	4
---	---	---	---	---	---	---

1	5	0	2	<u>4</u>	3	<u>6</u>
---	---	---	---	----------	---	----------

1	<u>3</u>	0	2	4	<u>5</u>	6
---	----------	---	---	---	----------	---

1	<u>2</u>	0	<u>3</u>	4	5	6
---	----------	---	----------	---	---	---

...

4

Insertion Sort

The items of the array are considered one at a time, and each new item is inserted into the appropriate position relative to the previously sorted items.

5

Insertion Sort

1	5	0	2	6	3	4
---	---	---	---	---	---	---

1	<u>0</u>	5	2	6	3	4
---	----------	---	---	---	---	---

<u>0</u>	1	5	2	6	3	4
----------	---	---	---	---	---	---

0	1	<u>2</u>	5	6	3	4
---	---	----------	---	---	---	---

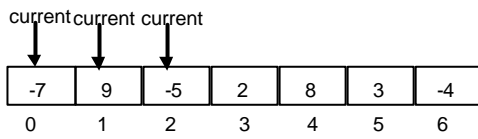
0	1	2	<u>5</u>	3	6	4
---	---	---	----------	---	---	---

0	1	2	<u>3</u>	5	6	4
---	---	---	----------	---	---	---

...

6

Linear Search

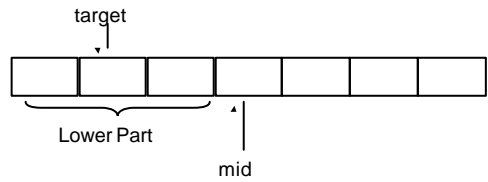


Target is -5

- Step through from the start until you find the target
- Numbers can be in any order
- Works for Linked Lists

7

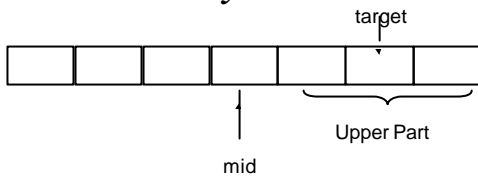
Binary Search



- Needs
 - List to be sorted.
 - To be able to do random accesses.

8

Binary Search

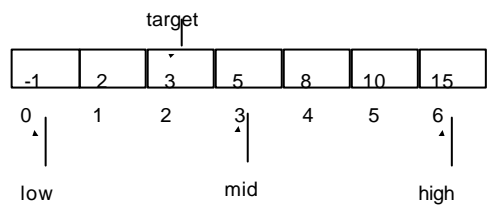


Other uses of binary search

- To find where a function is zero.
- Compute functions.
- Tree data structures.
- Data processing.
- Debugging code.

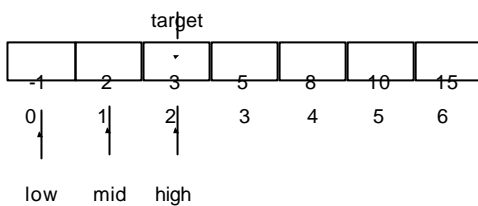
9

Binary Search



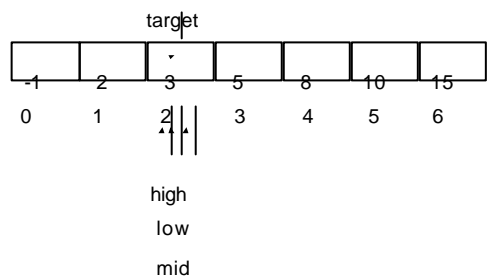
10

Binary Search




11

Binary Search



12



Recall:

Growth Rates / Complexity

- Constant
- Logarithmic
- Linear
- $n \log(n)$
- Quadratic
- Cubic
- Exponential
- $O(1)$
- $O(\log(n))$
- $O(n)$
- $O(n \log(n))$
- $O(n^2)$
- $O(n^3)$
- $O(2^n)$

13

Selection Sort

First find the largest element in the array and exchange it with the element in the last position, then find the second largest element in array and exchange it with the element in the second last position, and continue in this way until the entire array is sorted.

Complexity: ??

14

```
void selectionSort(float array[], int size)
{
    int j, k;
    int maxPosition;
    float tmp;

    for (k = size-1; k > 0; k--) {
        maxPosition = 0;

        for (j = 1; j <= k; j++) {
            if (array[j] > array[maxPosition]) {
                maxPosition = j;
            }
        }
        tmp = array[k];
        array[k] = array[maxPosition];
        array[maxPosition] = tmp;
    }
}
```

Find the maximum in a list

Complexity: $O(n^2)$ 15

Insertion Sort

The items of the array are considered one at a time, and each new item is inserted into the appropriate position relative to the previously sorted items.

Complexity: ??

16

```
void insertionSort(float array[], int size)
{
    int j, k;
    float current;

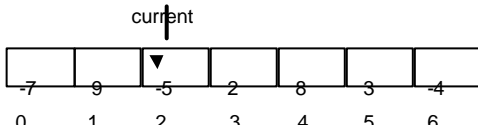
    for (k = 1; k < size; k++)
    {
        current = array[k];
        j = k;

        while (j > 0 && current < array[j-1]) {
            array[j] = array[j-1];
            j--;
        }

        array[j] = current;
    }
}
```

Complexity: $O(n^2)$ 17

Linear Search



- Step through from the start until you find the target
- Numbers can be in any order
- Works for Linked Lists

Complexity: ??

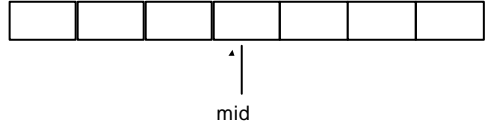
18

Linear Search

```
int linearSearch(float array[], int size, int target)
{
    int i;
    for (i = 0; i < size; i++)
    {
        if (array[i] == target)
        {
            return i;
        }
    }
    return -1;
}
```

Complexity: $O(n)$ 19

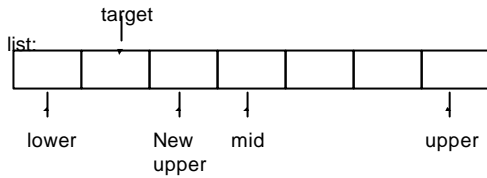
Binary Search



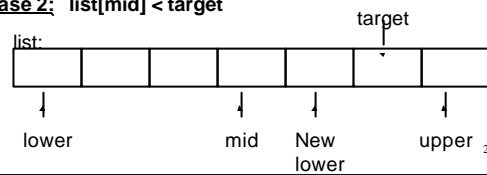
- Needs
 - List to be sorted.
 - To be able to do random accesses.
- Complexity: ??

20

Case 1: target < list[mid]



Case 2: list[mid] < target



21

```
int binarySearch(float array[], int size, int target)
{
    int lower = 0, upper = size - 1, mid;
    while (lower <= upper) {
        mid = (upper + lower)/2;
        if (array[mid] > target)
        {
            upper = mid - 1;
        }
        else if (array[mid] < target)
        {
            lower = mid + 1;
        }
        else
        {
            return mid;
        }
    }
    return -1;
}
```

Complexity: $O(\log(n))$ 22

The section where the target could be found halves in size each time

Comparison

	Array	Linked List
Selection Sort	✓	✓
Insertion Sort	✓	✓
Linear Search	✓	✓
Binary Search	✓	✗

23

Revision

- Selection Sort
- Insertion Sort
- Linear Search
- Binary Search

Revision: Reading

- Kruse 6, 7.1, 7.2
- Standish 6, 13.5
- Langsam 6.1, 6.4, 7.1
- Deitel & Deitel 6.6, 6.8

Preparation

- Next lecture: Recursion
- Read Section 3.2 and 3.4 in Kruse et al.

24