

**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	4	3	6
---	---	---	---	---	---	---

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

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

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

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

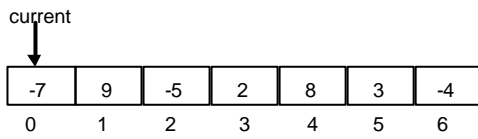
0	1	2	5	3	6	4
---	---	---	---	---	---	---

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

...

6

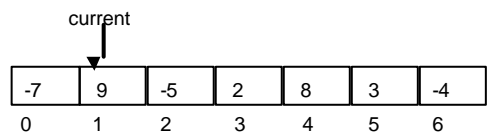
### Linear Search



- Target is -5

7

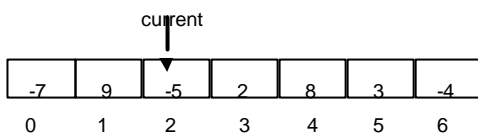
### Linear Search



- Target is -5

8

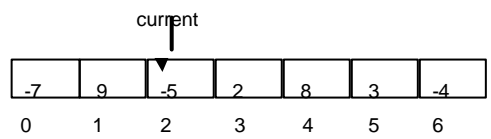
### Linear Search



- Target is -5

9

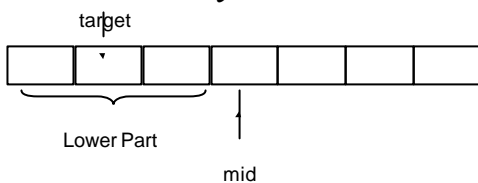
### Linear Search



- Numbers can be in any order.
- Works for Linked Lists.

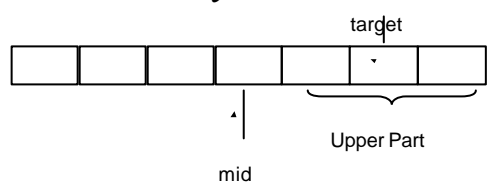
10

### Binary Search



11

### Binary Search



12

### Binary Search

A horizontal array of seven empty boxes. An arrow labeled 'target' points to the fifth box. An arrow labeled 'mid' points to the third box. A bracket under the last three boxes is labeled 'Upper Part'.

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

13

### Binary Search

A horizontal array of seven boxes containing the values -1, 2, 3, 5, 8, 10, 15. Below the boxes are indices 0 through 6. An arrow labeled 'target' points to the value 3 at index 2. An arrow labeled 'low' points to index 0. An arrow labeled 'mid' points to index 3. An arrow labeled 'high' points to index 6.

14

### Binary Search

A horizontal array of seven boxes containing the values -1, 2, 3, 5, 8, 10, 15. Below the boxes are indices 0 through 6. An arrow labeled 'target' points to the value 3 at index 2. Arrows labeled 'low', 'mid', and 'high' point to indices 0, 1, and 2 respectively.

15

### Binary Search

A horizontal array of seven boxes containing the values -1, 2, 3, 5, 8, 10, 15. Below the boxes are indices 0 through 6. An arrow labeled 'target' points to the value 3 at index 2. Arrows labeled 'high', 'low', and 'mid' point to indices 2, 1, and 2 respectively.

16

### Other Uses of Binary Search

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

17

### Recall: Growth Rates / Complexity

<ul style="list-style-type: none"> <li>• Constant</li> <li>• Logarithmic</li> <li>• Linear</li> <li>• <math>n \log(n)</math></li> <li>• Quadratic</li> <li>• Cubic</li> <li>• Exponential</li> </ul>	<ul style="list-style-type: none"> <li>• <math>O(1)</math></li> <li>• <math>O(\log(n))</math></li> <li>• <math>O(n)</math></li> <li>• <math>O(n \log(n))</math></li> <li>• <math>O(n^2)</math></li> <li>• <math>O(n^3)</math></li> <li>• <math>O(2^n)</math></li> </ul>
--	---

18

### *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.

19

### *Find where the Maximum is.*

```

/*
 * Find the position of the maximum in
 * list[0],...,list[k]
 */

maxPosition = 0;

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

```

20

```

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;
    }
}

```

21

### *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.

22

```

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

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

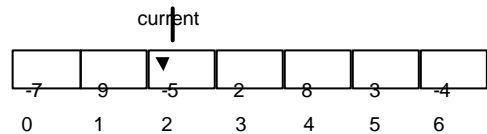
        while (j > 0 && current < array[j-1]) {
            array[j] = array[j-1];
            j--;
        }

        array[j] = current;
    }
}

```

23

### *Linear Search*



- Numbers can be in any order.
- Works for Linked Lists.

24

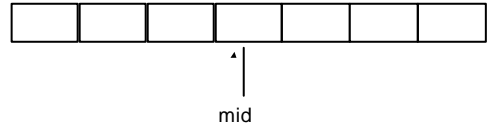
### Linear Search

```
int linearSearch(float array[], int size, int target)
{
    int i;

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

25

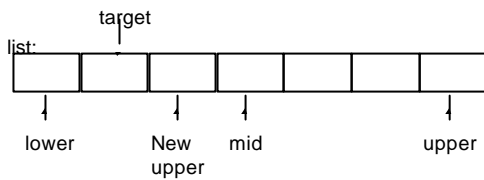
### Binary Search



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

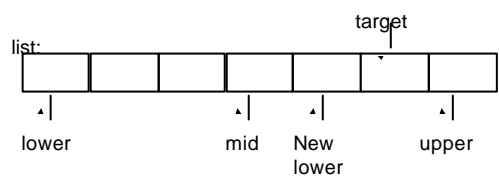
26

#### Case 1: target < list[mid]



27

#### Case 2: list[mid] < target



28

```
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;
}
```

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

29

### Comparison

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

30

***Revision***

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

31

***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.

32