

CS1100

Introduction to Programming

Searching and Structures

Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Course Material – SD, SB, PSK, NSN, DK, TAG – CS&E, IIT M

1

Searching for Elements

- Given an array of marks and names, is there someone who got X marks?
- If X occurs in the Marks array, return the index of the position where it occurs.
- If the numbers are randomly arranged, there is no option but to scan the entire array to be sure.
- Would the task be simpler if the marks were arranged in sorted order?
 - Like finding a word in a dictionary

SD, PSK, NSN, DK, TAG – CS&E, IIT M

2

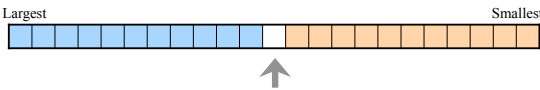
Searching in a Sorted Array

- Given an array of marks and names sorted in *descending* order of marks, is there someone who got X marks?
- If X is high (say 92/100), one could start scanning from the left.
- If X is low (say 47/100), one could scan the array right to left.
- But what if we do not know whether X is high or low?

SD, PSK, NSN, DK, TAG – CS&E, IIT M

3

Divide and Conquer



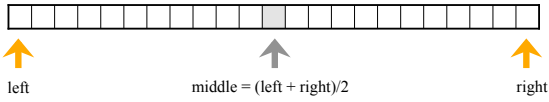
- Look at the middle element
- If $\text{array}[\text{middle}] = X$, done
- If $\text{array}[\text{middle}] > X$, look *only in the right part*
- Else look for the number *only in the left part*
- The problem is reduced into a smaller problem
 - new problem is half the size of the original one*
- Recursively apply this strategy*

SD, PSK, NSN, DK, TAG – CS&E, IIT M

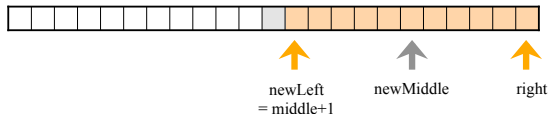
4

Divide and Conquer

- Two indexes define the range of searching



- If $\text{array}[\text{middle}] > X$ look *only in the right part*



SD, PSK, NSN, DK, TAG – CS&E, IIT M

5

Binary Search (also called Binary Chop)

- Starts with the full sorted array
 - left = 0 and right = N-1
- The range of search are the elements between left and right including $\text{array}[\text{left}]$ and $\text{array}[\text{right}]$
- Search terminates if $\text{right} < \text{left}$
- Otherwise
 - If $(\text{array}[\text{middle}] == X)$ return middle
 - If $(\text{array}[\text{middle}] < X)$ left = middle + 1
 - Else right = middle - 1

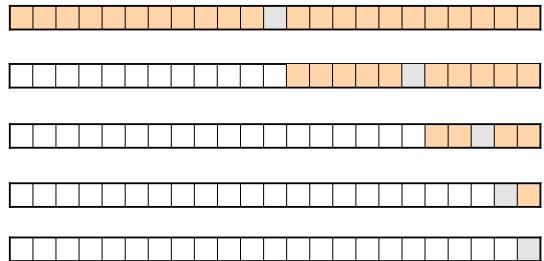
SD, PSK, NSN, DK, TAG – CS&E, IIT M

6

Binary Search (bsearch in stdlib.h – elements increasing order)

```
int BinarySearch(int value, int array[], int n){
    int left = 0, right = n-1;
    while (left <= right){
        middle = (left+right)/2;
        if(array[middle] == value) return middle;
        if(array[middle] < value) left = middle +1;
        else right = middle -1;
    }
    return INFINITY; /*calling function must interpret this correctly! */
}
```

Complexity of Binary Search



After each inspection the array reduces by half. For an array of size N there are about $\log_2 N$ inspections in the worst case.

Complexity: Recurrence Equation

- Let n be the size of the array
- Let the function $T(n)$ represent the time complexity of solving the problem of size n
- In each call the problem is reduced by half
 - one comparison in each call
- $T(n) = 1 + T(n/2)$
- Solving this gives us the complexity $\log_2(n)$

n	128	256	512	1024	2048	4096	...		
log n	7	8	9	10	11	12	...		

$$T(n) = 1 + T(n/2)$$

$$T(n/2) = 1 + T(n/4)$$

$$T(n/4) = 1 + T(n/8)$$

$$\vdots$$

$$T(2) = 1 + T(1)$$

$$= 1 + 1$$

Finding Student with X marks

```
char *findStudent (int x, char names[][COL], int marks[],
                  int n)
{
    int index = BinarySearch(x, marks[], n);
    if (index == n) return "no such student";
    else return names[index];
}
return NULL;
```

Above function assumes that names are corresponding to marks, and marks are in decreasing order.

Marks and Names

- We kept Marks in an integer array and Names in a corresponding two dimensional array
 - could keep Names in an array of pointers
 - memory reserved as per requirements
- The connection between the two arrays was via the shared index
- Can we keep them in the same array?
 - one option – keep first three characters for marks and convert them while processing
 - a better option – use structures

Structures

- Collection of one or more variables, possibly of different types, grouped together under a single name for easy handling.
- For example - a structure which represents a point in a two dimensional plane

```
struct point {
    int x;
    int y;
};
```

A mechanism for defining compound data types

By itself it reserves no storage

Point in 2D → 2 Integers

- Different ways of declaring structure variables

```
struct point{
    int x;
    int y;
} point1, point2;
```

```
struct point point1, point2;
```

```
struct point point1 = { 3 , 2 };
```

SD, PSK, NSN, DK, TAG – CS&E, IIT M

13

Marks and Names

```
struct student{
    char *name;
    int mark;
}s1, s2;

struct student s1, s2;
struct student s1 = { "Ramesh" , 79 };
```

name could itself be a struct made up of first name, middle name and last name...
Nested structures are allowed

SD, PSK, NSN, DK, TAG – CS&E, IIT M

14

A Rectangle

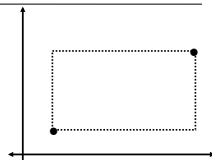
```
struct rectangle{
    struct point pt1;
    struct point pt2;
}rect1;
```

- Accessing points in the rectangle

```
rect1.pt1.x = 4;
rect1.pt1.y = 5;
```

Or

```
rect1.pt1 = { 4, 5 };
```



SD, PSK, NSN, DK, TAG – CS&E, IIT M

15

Defining New Types

- 'typedef' keyword is used for creating new data types

- For example:

```
typedef int Age;
Age myAge = 99;
```

- typedef and Structures:

```
typedef struct point pointType;
pointType point1, point2;
```

- This is equivalent to: `struct point point1, point2;`

SD, PSK, NSN, DK, TAG – CS&E, IIT M

16

Operations on Structures

- Structures may be copied by assignment statement
- The address of the structure (use &) can be passed to functions and can be returned by functions
 - one can pass an entire structure
 - one can pass some components of a structure
 - one can pass a pointer to a structure
- Structures may not be compared

SD, PSK, NSN, DK, TAG – CS&E, IIT M

17

Functions and Structures

- Structure as function argument

```
int isOrigin(pointType pt){
    if(pt.x == 0 && pt.y == 0)
        return 1;
    else
        return 0;
}
```

SD, PSK, NSN, DK, TAG – CS&E, IIT M

18

Structures and Functions

- Structure as return type

```
pointType makePoint(int x, int y){
    pointType temp;
    temp.x = x;
    temp.y = y;
    return temp;
}
```

Observe there is no confusion between the two occurrences of x and y

SD, PSK, NSN, DK, TAG - CS&E, IIT M

19

A Screen and its Centre Point

```
struct rectangle screen;
```

```
struct point middle;
```

```
struct point makePoint(int, int);
```

```
screen.pt1 = makePoint(0, 0);
```

```
screen.pt2 = makePoint(XMAX, YMAX);
```

```
middle =
```

```
    makePoint((screen.pt1.x+screen.pt2.x)/2,
              (screen.pt1.y+screen.pt2.y)/2);
```

SD, PSK, NSN, DK, TAG - CS&E, IIT M

20

Adding Two Points

```
/* addPoints: add two points */
```

```
struct point addPoints(struct point p1, struct point p2)
```

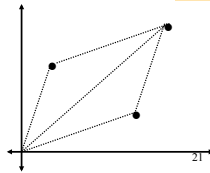
```
{    p1.x += p2.x;
```

```
    p1.y += p2.y;
```

```
    return p1;}

```

Note that the local changes to p1 would not affect the point p1; *pass by value*



SD, PSK, NSN, DK, TAG - CS&E, IIT M

21