Computer Programming & Problem Solving

CS100

Mrs Sanga G. Chaki
Department of Computer Science and Engineering
National Institute of Technology, Goa
May, 2023

Contents

1. Unary Operator: sizeof()
2. ceil() and floor() functions
3. Searching
    a) Linear search
    b) Binary
4. Sorting
    a) Bubble sort

sizeof

1. **Unary operator which can be used to compute the size of its operand.**

```c
#include <stdio.h>
int main() {
    int i;
    int a[] = {1,2,3,4};
    float f;
    char c;
    printf("The sizes of different dataypes in this machine:\n");
    printf("The size of int = %d\n", sizeof(int));
    printf("also the size of int = %d\n", sizeof(i));
    printf("The size of float = %d\n", sizeof(f));
    printf("The size of char = %d\n", sizeof(c));
    printf("The size of array a = %d\n", sizeof(a));
    return 0;
}
```

sizeof

Output

```
/tmp/rLgJi0f54k.o
The sizes of different dataypes in this machine:
The size of int = 4
also the size of int = 4
The size of float = 4
The size of char = 1
The size of array a = 16
```

1. **Try this in your machine**

ceil() and floor()

1. Standard library functions to roundoff float values to integers
2. The ceil function in C returns the nearest integer greater than the provided argument – argument is a float.
3. floor() function returns the nearest integer smaller than the argument

ceil() and floor()

```c
#include <stdio.h>
#include <math.h>

int main(){
    double num = 8.33;
    int r1, r2;
    r1 = ceil(num);
    printf("Ceiling integer of %.2f = %d", num, r1);
    r2 = floor(num);
    printf("\nFloor integer of %.2f = %d", num, r2);
    return 0;
}
```

Output

```
/tmp/gGn0GYZUCQ.o
Ceiling integer of 8.33 = 9
Floor integer of 8.33 = 8
```

SEARCHING and SORTING

Searching

1. Check if a given element occurs in the array.
2. Two ways:
    a) If the array elements are unsorted - Linear search
    b) If the array elements are sorted - Binary search

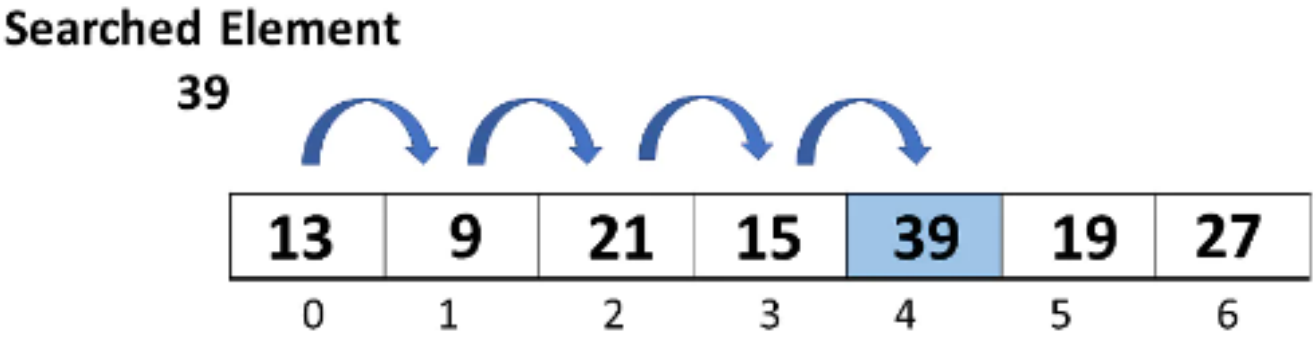
Linear Search

1. **Basic idea:**

   a) **Start at the beginning of the array.**

   b) **Inspect elements one by one to see if it matches the key (the element being searched).**

Linear Search - Example



Linear Search - Example

```c
#include <stdio.h>

int main(){
    int arr[] = {2, 3, 4, 5, 7, 1, 10};
    int x = 4;
    int N = sizeof(arr) / sizeof(arr[0]);
    int i;
    int flag = 0;
    for (i = 0; i < N; i++){
        if (arr[i] == x){
            printf("\nElement is present at index %d", i);
            flag += 1;
//          break;
        }
        else
            continue;
    }
    if(!flag)
        printf("Element is not present in array");
    return 0;
}
```

3 new terms of Algorithmic Performance

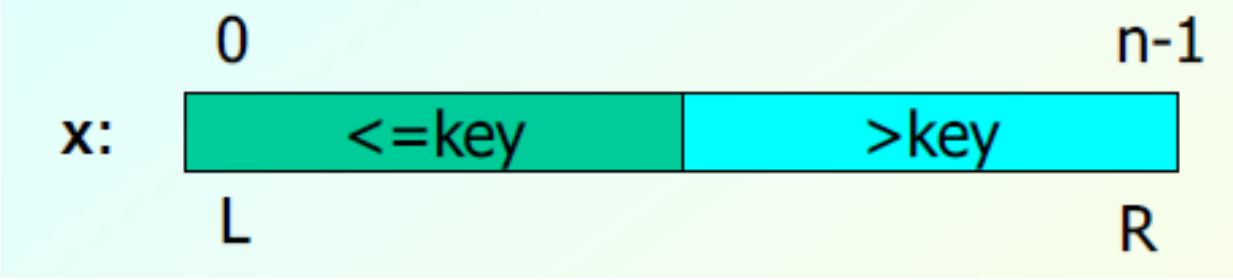
1. If there are *n* elements in the array:
    a) Best case: match found in first element (1 search operation)
    b) Worst case: no match found, or match found in the last element (n search operations)
    c) Average case: (n + 1) / 2 search operations

Binary Search

1. Basic idea: Binary search works if the array is sorted.
    a) Look for the target in the middle.
    b) If you don't find it, you can ignore half of the array, and repeat the process with the other half.
2. In every step, we reduce the number of elements to search in by half.

Binary Search

1. Strategy: Find split between values larger and smaller than key
2. Situation while searching: Initially L and R contains the indices of first and last elements.
3. Then, look at the element at index [(L+R)/2].
4. Move L or R to the middle depending on the outcome of test.



Binary Search - Example

Binary Search - Example

```c
// Binary Search in C
#include <stdio.h>
int main(void) {
  int array[] = {3, 4, 5, 6, 7, 8, 9};
  int n = sizeof(array) / sizeof(array[0]);
  int x = 8; //to be searched
  int mid, high, low, res = -1;
  low = 0;
  high = n-1;
  // Repeat until the pointers low and high meet each other
  while (low <= high) {
    mid = low + (high - low) / 2;
    if (array[mid] == x)
      res = mid;
    if (array[mid] < x)
      low = mid + 1;
    else
      high = mid - 1;
  }
  if (res == -1)
    printf("Not found");
  else
    printf("Element is found at index %d", res);
  return 0;
}
```

Why use Binary Search?

1. Suppose that the array x has 1000 elements.
2. In Linear search – If key is a member of x, it would require 500 comparisons on the average.
3. In Binary search
   a) after 1st compare, left with 500 elements.
   b) after 2nd compare, left with 250 elements.
   c) after at most 10 steps, you are done.
   d) If there are n elements in the array, number of searches required in the worst case: $log_2 n$

Sorting

1. Basic Problem: Given an array

$$x[0], x[1], \ldots, x[size-1]$$

reorder entries so that

$$x[0] <= x[1] <= \ldots <= x[size-1]$$

So that, the array is in non-decreasing or non-increasing order.

2. Example: If original list: 10, 30, 20, 80, 70, 10, 60, 40, 70

a) Sorted in non-decreasing order: 10, 10, 20, 30, 40, 60, 70, 70, 80

b) Sorted in non-increasing order: 80, 70, 70, 60, 40, 30, 20, 10, 10

Bubble Sort

1. The sorting process proceeds in several passes.
2. In every pass, we go on comparing neighboring pairs, and swap them if out of order.
3. If we are sorting in ascending order, in every pass, the largest of the elements under consideration will bubble to the top (i.e., the right).
4. Number of comparisons: n(n-1)/2, if there are n elements in the array.

Bubble Sort – Worked out example

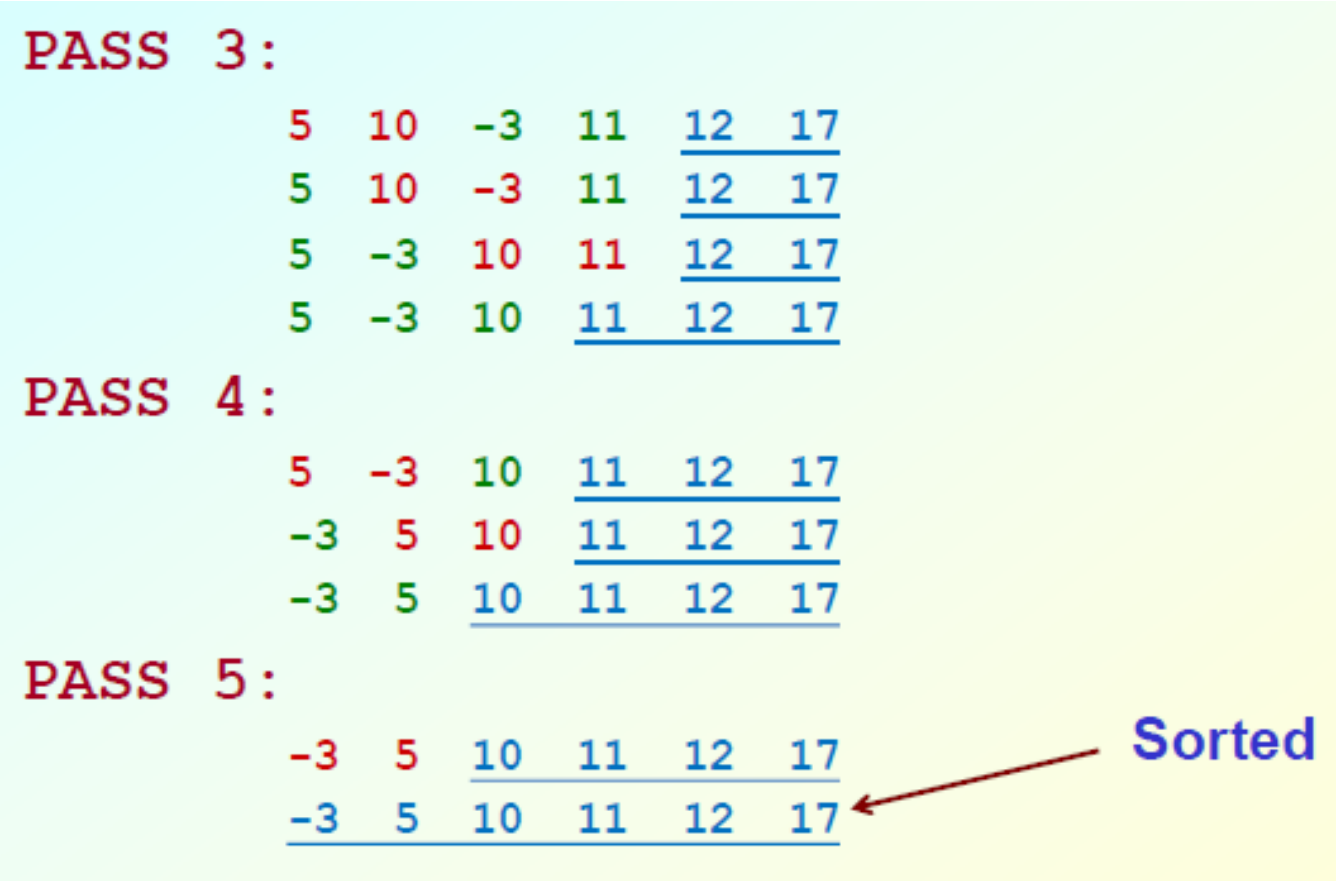
PASS 1:

10	5	17	11	-3	12
5	10	17	11	-3	12
5	10	17	11	-3	12
5	10	11	17	-3	12
5	10	11	-3	17	12
5	10	11	-3	12	17

PASS 2:

5	10	11	-3	12	17
5	10	11	-3	12	17
5	10	11	-3	12	17
5	10	-3	11	12	17
5	10	-3	11	12	17

Bubble Sort – Worked out example



Homework: Code

Bubble Sort – Code

```c
#include <stdio.h>
int main() {//Bubble sort
   int array[] = {10, 5, 17, 11, -3, 12};//array to be sorted
   int size = 5, step, i, temp;
   for(step= 0;step<size - 1;step++) {// loop to access each array element
     for(i=0; i<size - step - 1;i++) {// loop to compare array elements
       if(array[i] > array[i + 1]) {// compare two adjacent elements
         temp = array[i]; // swap if elements not in the intended order
         array[i] = array[i + 1];
         array[i + 1] = temp;
       }//end if
     }//end inner for loop
   }//end outer for loop
   printf("Sorted Array in Ascending Order:\n");
   for (i = 0; i < size; i++)
     printf("%d  ", array[i]);
   return 0;
}
```

Useful Library Functions

1. math.h
   a) Floor
   b) Ceil
   c) Log
   d) Mod
   e) Sqrt
   f) Pow
2. string.h
   a) Strlen
   b) Strcpy
   c) Strcat
   d) strcmp
1. stdlib.h
   a) rand
   b) exit
   c) Malloc
   d) free
2. stdio.h
   a) Printf
   b) Scanf
   c) Getc
   d) Putc
   e) Fopen
   f) Fclose