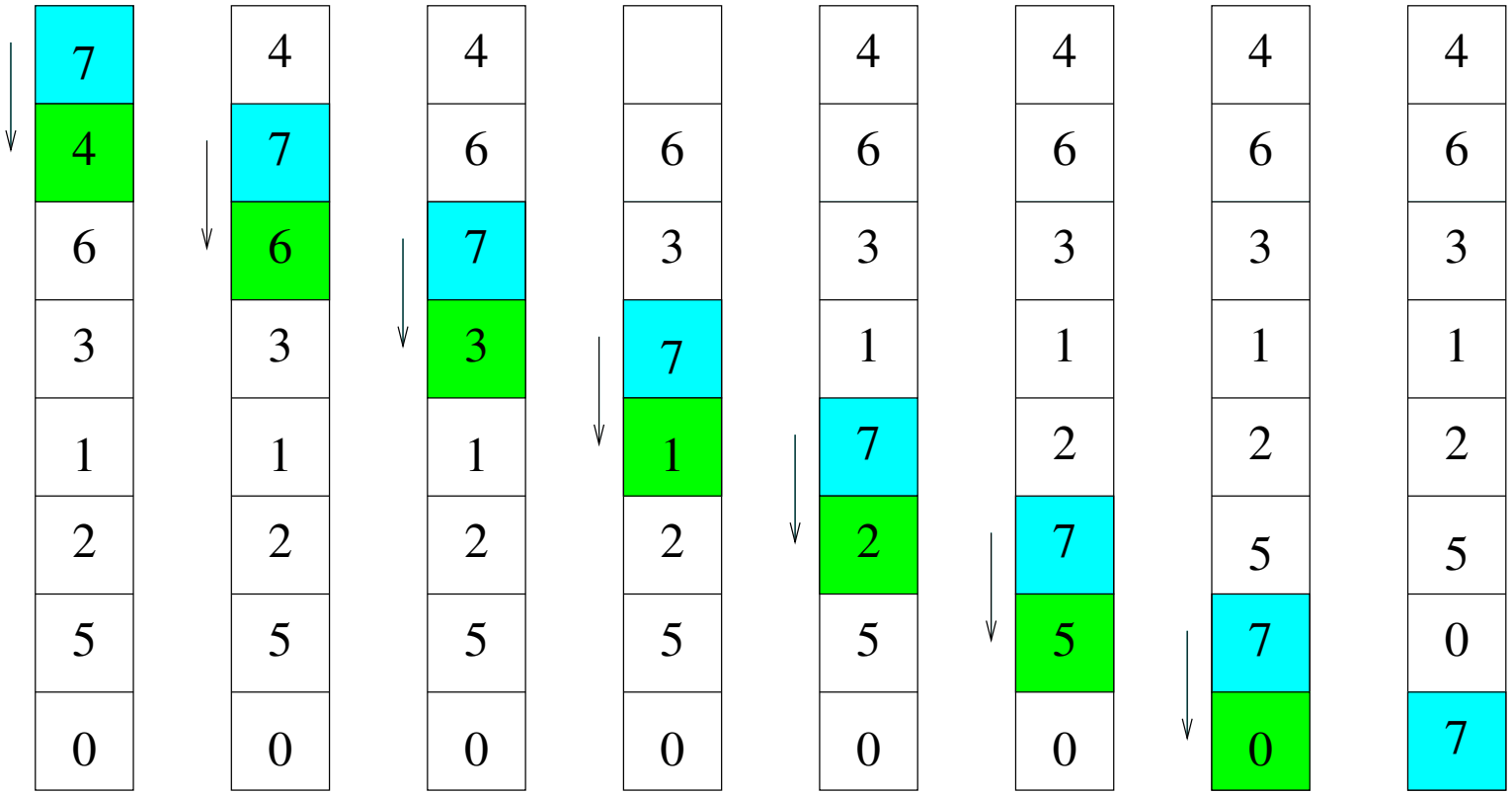
Sorting

Bubble Sort

- Bubble sort **sinks** heaviest element to bottom causing the lighter elements to **bubble** up.
- Bubble step is as follow:
  1. Start from the first element in the array, compare adjacent pair of elements.
  2. Swap if required to push the lighter of the pair one position up.
  3. Repeat the comparison of next pair until 1st and 2nd element have been compared.
- Sorting is accomplished by repeating **bubble** operation n times.

Sorting

Bubble Step



Sorting

Code for Bubble Sort

```
void bubbleSort(int a[], int n) {
    int i, j, temp;
    for (i = n-1; i > 0; i--) // Execute bubble operation n times
        for (j = 1 ; j <=  i; j++) // Bubble operation
            if (a[j-1] > a[j]) {
                temp = a[j-1];
                a[j-1] = a[j];
                a[j] = temp;
            }
}
```

Sorting

Putting All Together

```
#include <time.h>
#include <stdio.h>
#define N 10
int main() {
    int a[N];
    int n = sizeof(a)/sizeof(a[0]);
    generate(a, n);
    printf("Unsorted input \n");
    printArray(a, n);
    bubbleSort(a, n);
    printf("Sorted output \n");
    printArray(a, n);
}
```

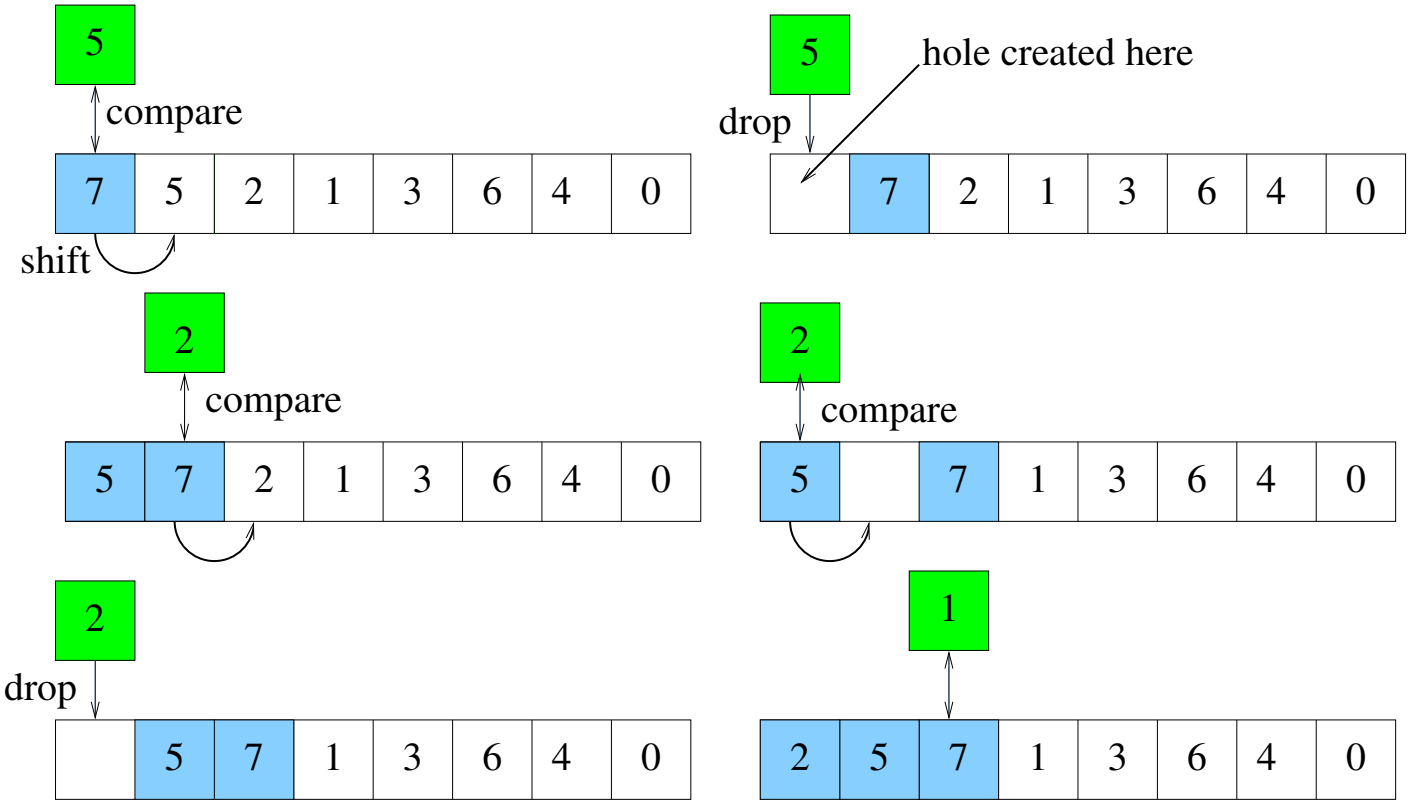
Sorting

Insertion Sort

- Based on the act of picking cards and arranging the hand in sorted order.
- Initially, only the first element of the array is in sorted order.
- The input element is compared element-wise with sorted part of the array.
- If the input is smaller than the current element of the sorted part, consider the next element of sorted part.
- Else position for the input element is found.

Sorting

Insertion Step



Sorting

Code for Insertion

- Insertion step should be performed by each element starting from first.
- Insertion step at stage i is as follow:

```
/* Insertion step assuming a[0, ..., i-1] forms sorted part */

x = a[i]; // Save next unsorted element
j = i-1;  // Get index of last element of sorted part

while (x < a[j] && j >= 0) {
    a[j+1] = a[j];  // Shift the larger element to right
    j--;            // Get index of next largest in sorted part
}
a[j+1] = x;         // Drop input element in current position
```

Sorting

Insertion Sort

```
insertionSort(int a[], int n) {
    int i, j, x;
    for (i = 1; i < n; i++) {
        x = a[i];
        j = i-1;
        while (x < a[j] && j >= 0) {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = x;
    }
}
```

Sorting

Putting All Together

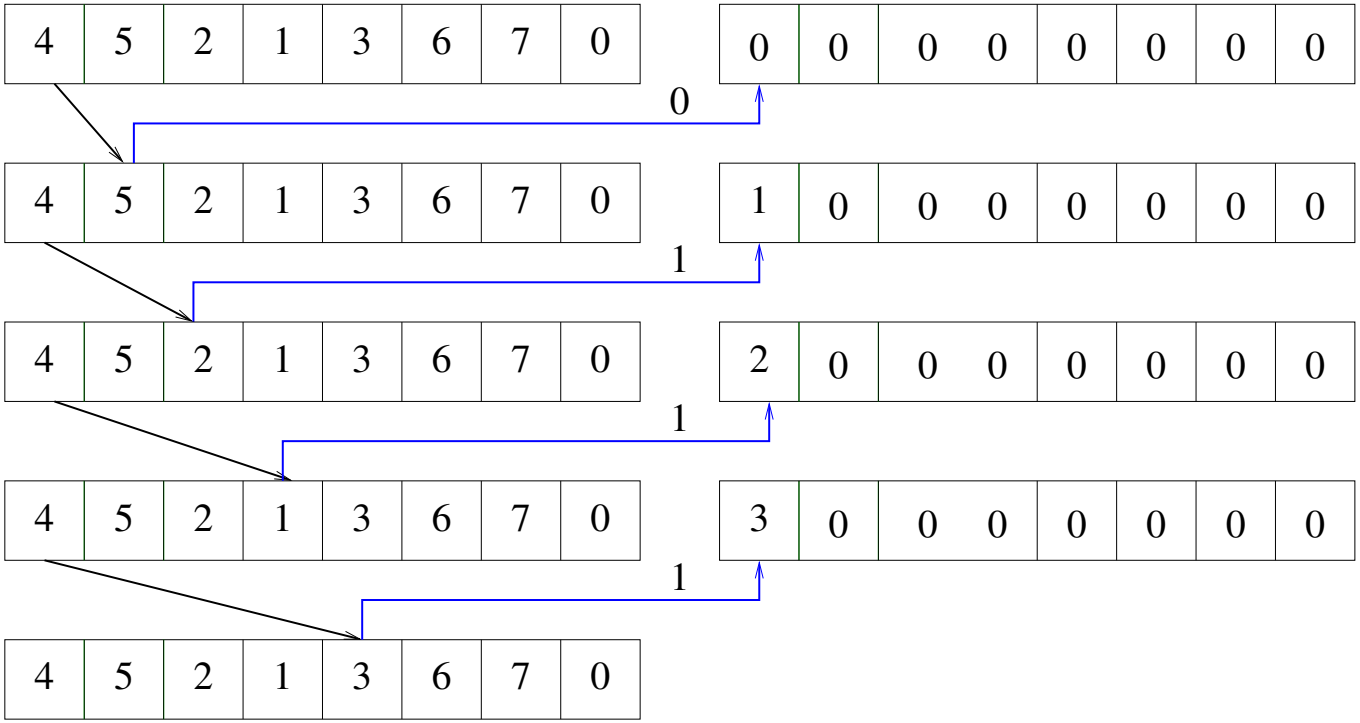
```
#include <stdio.h>
#include <time.h>
#define N 10
int main() {
    int a[N];
    int n = sizeof(a)/sizeof(a[0]);

    generate(a, n);
    printf("\nUnsorted input \n");
    printArray(a, n);
    insertionSort(a, n);
    printf("\n\nSorted output \n");
    printArray(a, n);
}
```

Sorting

Counting Sort

For each element count the number of other elements which are greater.



Sorting

Counting Sort

```
void countingSort(int a[], int n) {
    int j, k, b[n], cnt[n];

    for (j = 0; j < n; j++)
        cnt[j] = 0;    // Counters initialized

    for (j = 0; j < n; j++) {
        for (k = 0; k < n; k++) {
            if (a[j] < a[k])  // a[k] shd appear after a[j]
                cnt[k]++;
            if (a[j] == a[k] && k > j) // maintain relative pos.
                cnt[k]++;              // of equal elements
        }
    }
    /* Remaining part of function appears in next slide */
}
```

Sorting

Counting Sort (contd.)

```
/* Placing a[j] in position a[cnt[j]] completes sorting. */

  for (j = 0; j < n; j++)
     b[cnt[j]] = a[j];       // Use temporary array

  for (j = 0; j < n; j++)
     a[j] = b[j];            // Transfer back
```

Searching

Linear Search

- Another very frequently used operation.
- **Sequential search**: look at each element from the beginning and match with the target x.
  - Returns index of matched element if x is found.
  - Returns -1 if x is does not occur.

Searching

Code for Linear Search

```
int sequentialSearch(int a[], int n, int x) {
    int i;

    for (i = 0; i < n; i++)
        if (a[i] == x) break;
    if (i < n)
            return i;
        else
            return -1
}
```

Searching

Code for Linear Search

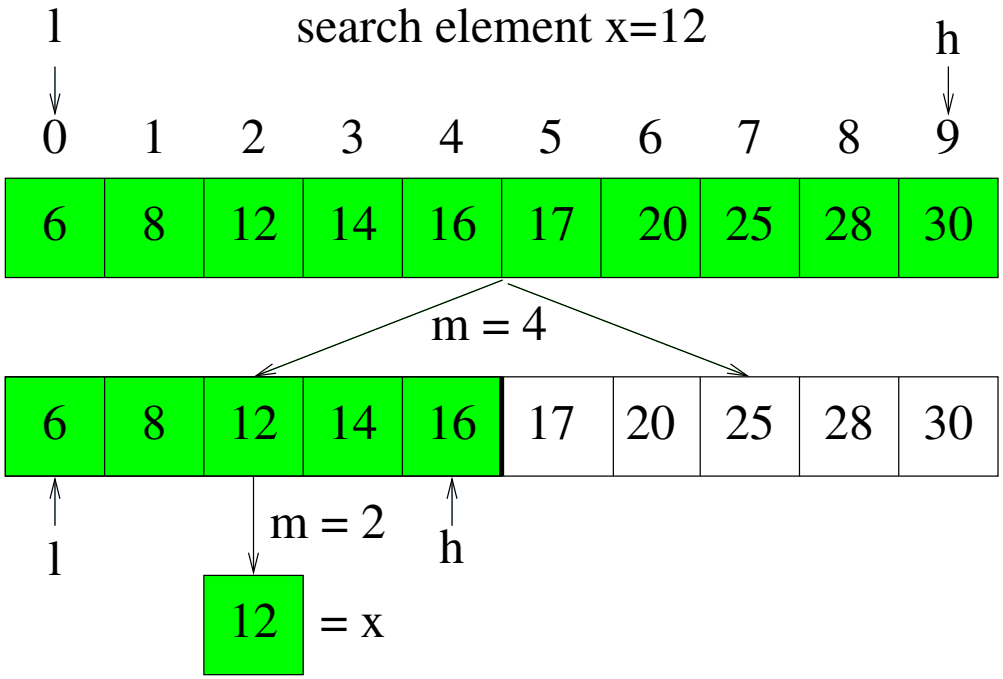
```
#define N 10
int main() {
    int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    int x, i;
    int n = sizeof(a)/sizeof(a[0]);

    printf("Enter target element: ");
    scanf("%d", &x);

    i = sequentialSearch(a, n, x);
    if (i == -1)
        printf("%d is not found!\n", x);
    else
        printf("%d found at position %d \n",x,i);
}
```

Searching

Binary Search



Searching

Binary Search

```
int binarySearch(int a[], int l, int h, int x) {
    int m;
    while (l <= h) {
        m = (l + (h - l)/2);
        if (a[m] == x)
            return m;
        else if (a[m] < x)
            l = m+1;
        else
            h = m-1;
    }
    return -1;
}
```

Searching

Binary Search

```
#include <stdio.h>
int main() {
    int a[N] = {0, 1, 2, 3, 5, 6, 8, 9, 11, 12};
    int n = sizeof(a)/sizeof(a[0]);
    int i, x;

    printf("Enter the number to be searched: ");
    scanf("%d", &x);
    printf("Input \n");
    printArray(a, n);

    i = binarySearch(a, 0, n-1, x);
    if (i == -1)
        printf("%d is not found!\n", x);
    else
        printf("%d found at position %d \n",x,i);
}
```