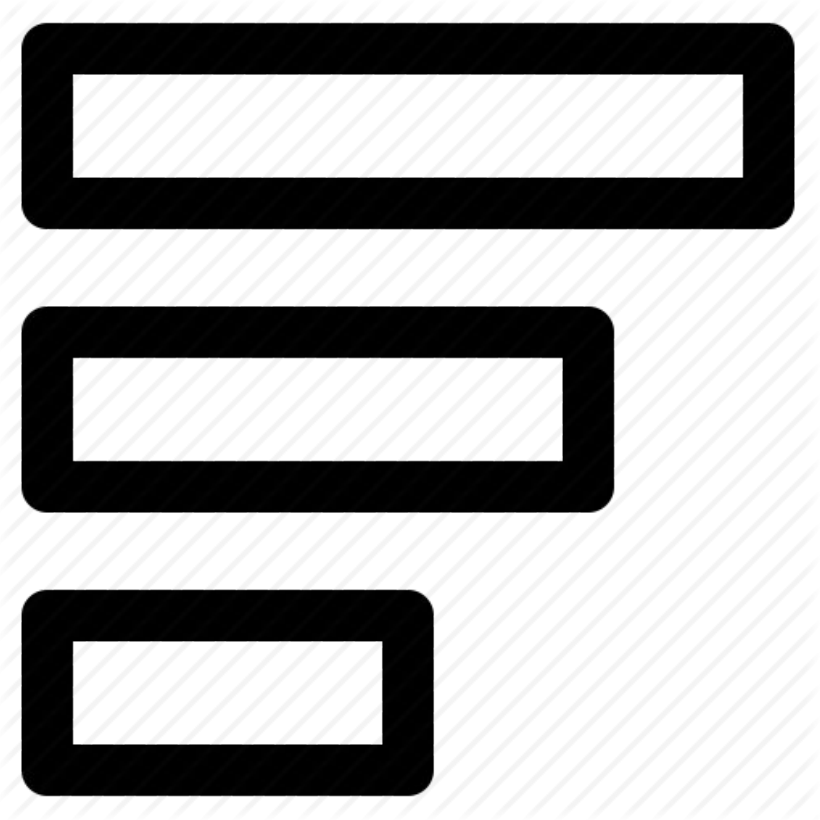
Algorithms and Data Structures

Team 1

- Humberto Bernal
- Joseph Peña

Contenido

- Bubble sort
- Selection sort
- Binary search
- Questions

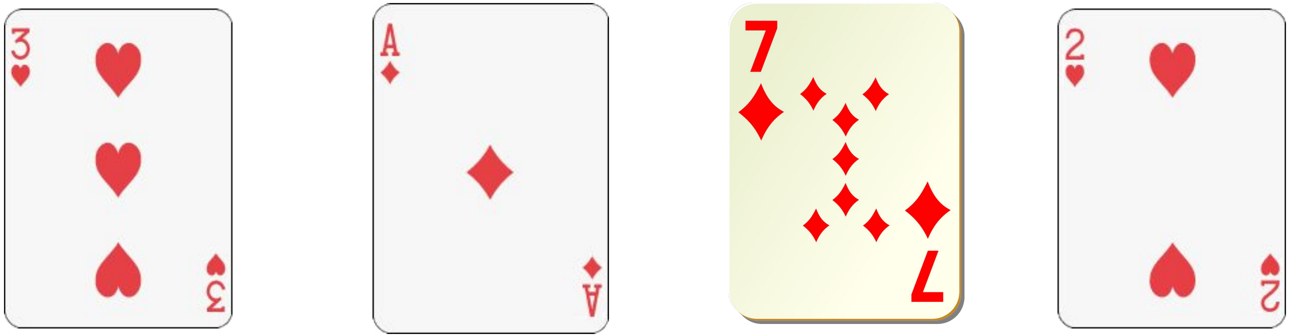


Bubble sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.[1]

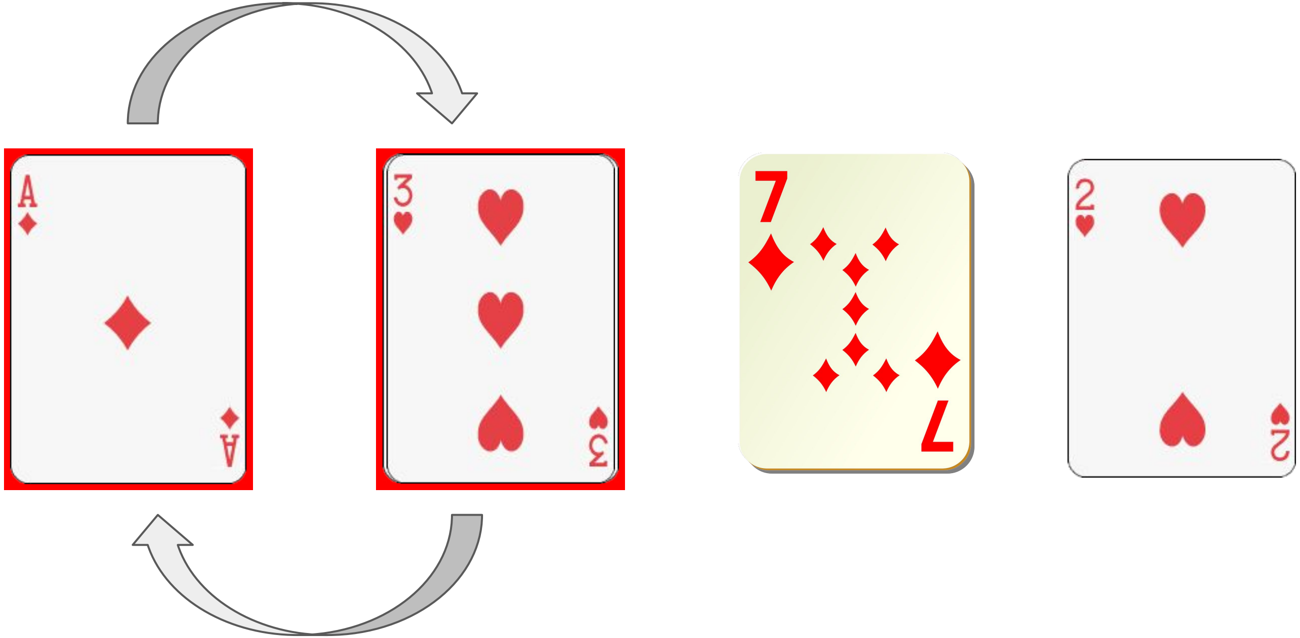
Example:

Given a set of unsorted cards:

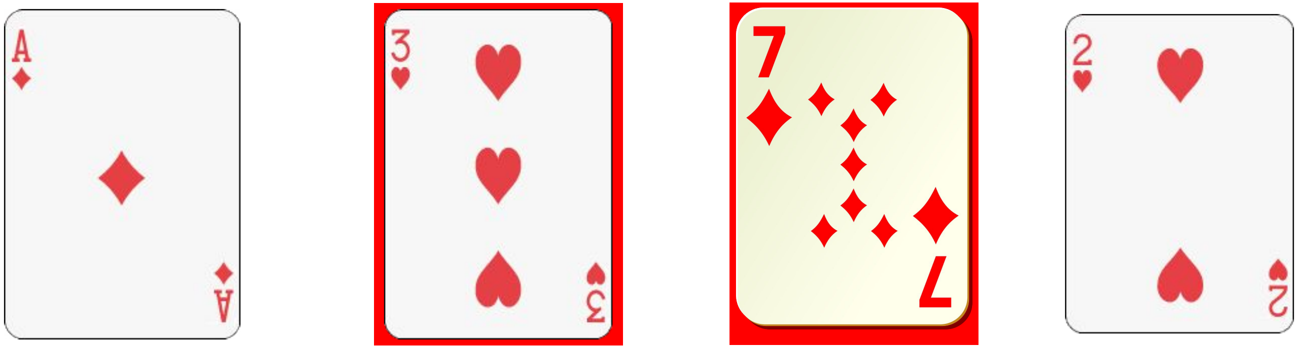


[1] https://www.geeksforgeeks.org/bubble-sort/

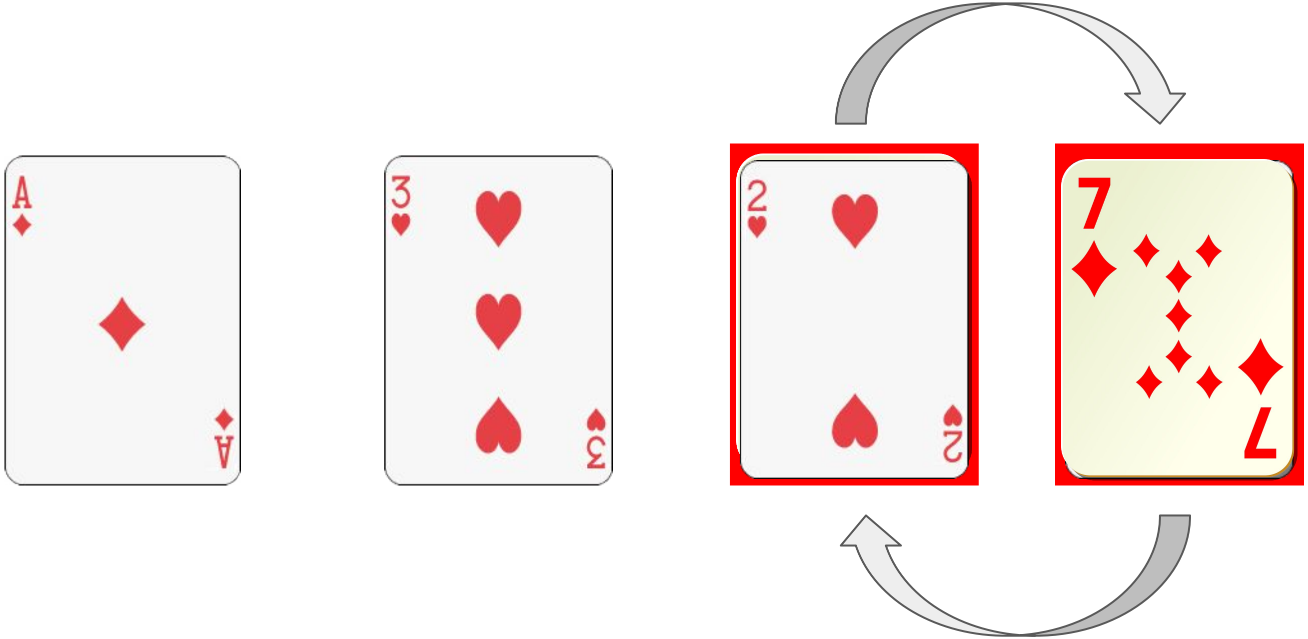
Bubble sort



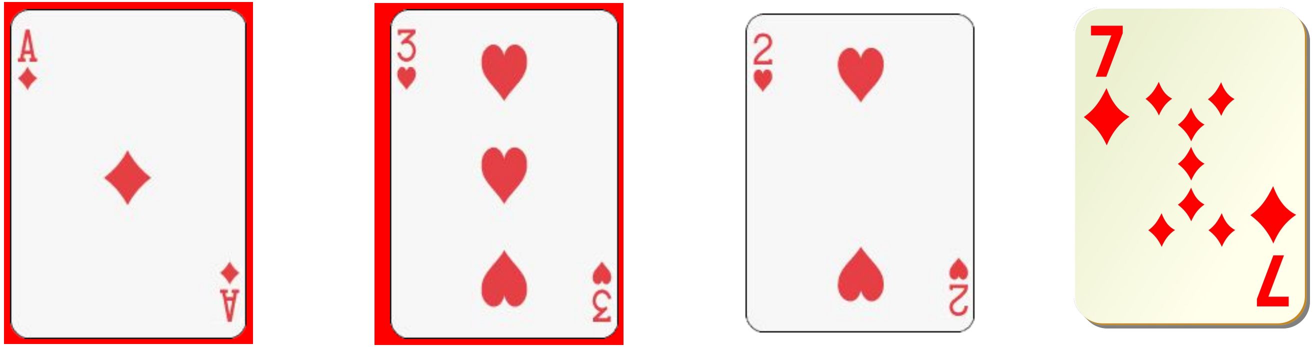
Bubble sort



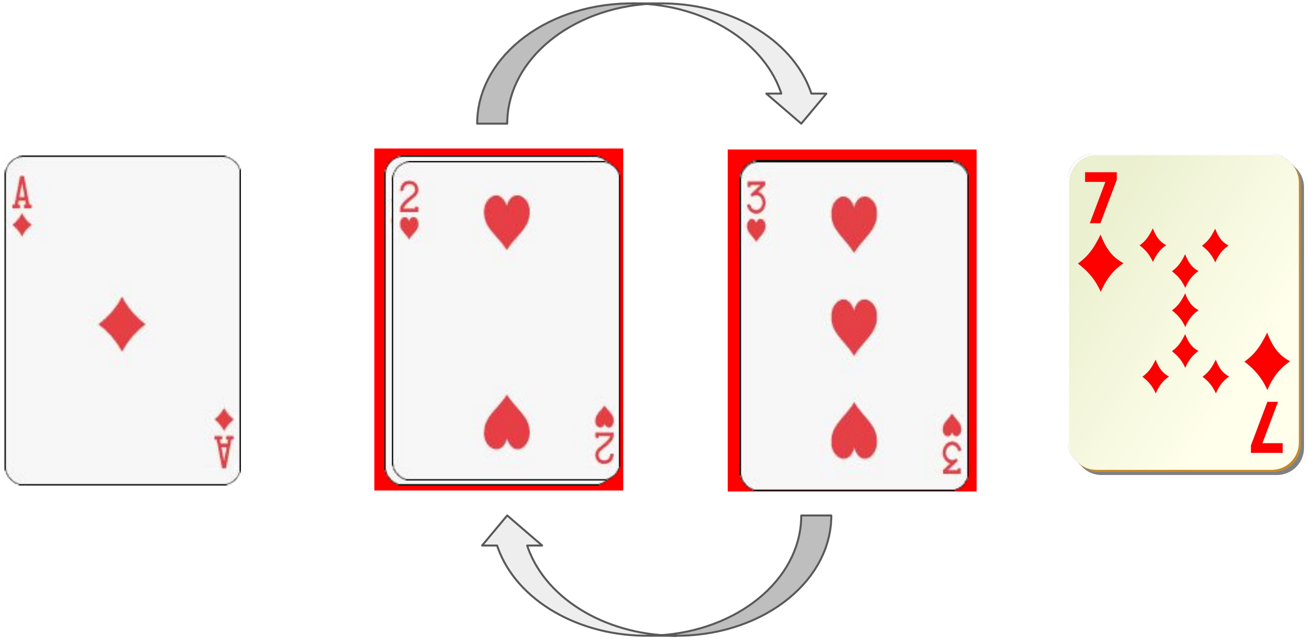
Bubble sort



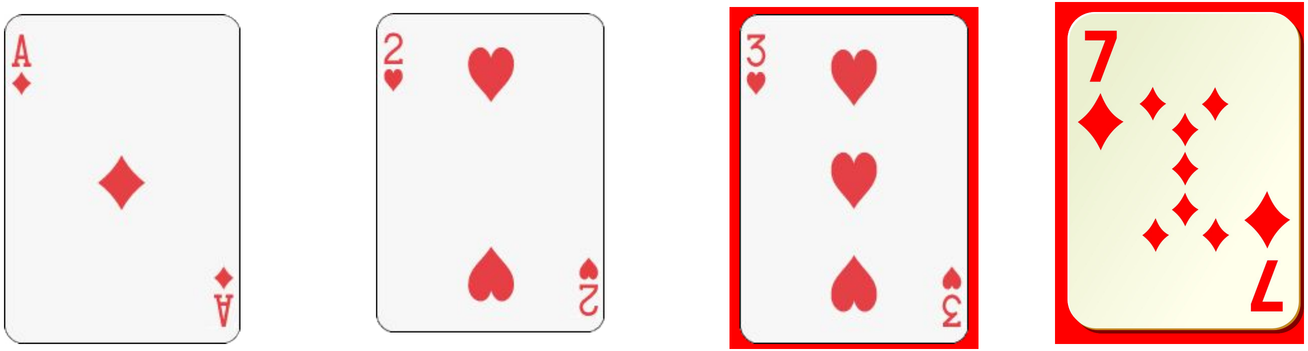
Bubble sort



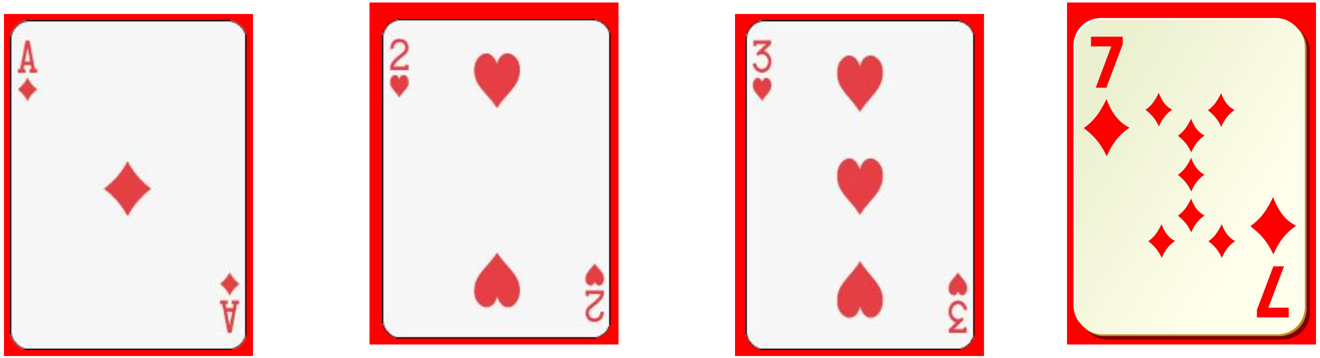
Bubble sort



Bubble sort



Bubble sort



Bubble sort

```
int main()
{
    array<int, arr_length> arr = get_random_arr();

    for (int i=0; i < arr_length; ++i)
        for (int j=0; j < arr_length-i; ++j)
            if (arr[j] > arr[j+1])
                swap_values(arr, j, j+1);

}
```

arr = 8, 9, 9, 2, 4

Iteración 1:
8, 9, 9, 2, 4
Iteración 2:
8, 9, 9, 2, 4
Iteración 3:
8, 9, 2, 9, 4
Iteración 4:
8, 9, 2, 4, 9
Iteración 5:
8, 9, 2, 4, 9
Iteración 6:
8, 9, 2, 4, 9
Iteración 7:
8, 2, 9, 4, 9
Iteración 8:
8, 2, 4, 9, 9
Iteración 9:
8, 2, 4, 9, 9
Iteración 10:
2, 8, 4, 9, 9
Iteración 11:
2, 4, 8, 9, 9
Iteración 12:
2, 4, 8, 9, 9
Iteración 13:
2, 4, 8, 9, 9
Iteración 14:
2, 4, 8, 9, 9
Iteración 15:
2, 4, 8, 9, 9

Bubble sort

```
int main()
{
    array<int, arr_length> arr = get_random_arr();

    for (int i=0; i < arr_length; ++i)
        for (int j=0; j < arr_length-i; ++j)
            if (arr[j] > arr[j+1])
                swap_values(arr, j, j+1);

}
```

Loop 1

Loop 2

Used space:

- Variables from ***swap*** function (These are deallocated when the function ends)
- i and j

Number of operations:

- Since the loop 1 is inside of loop2, the result is Loop1*Loop2= O(x^2)

Bubble sort: Summary

Computational cost:

- Time: x^2
- Space: 1
- Stability: yes

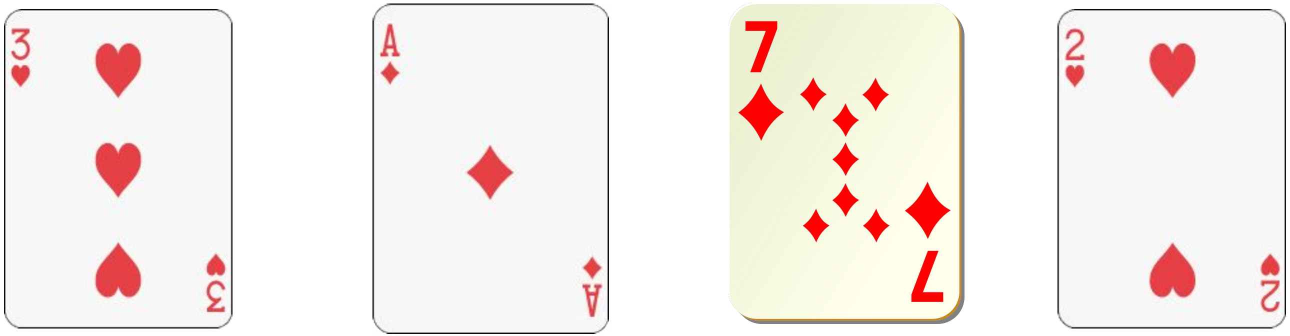
When to use it?

- When there is not enough program memory space
- When the length of the list/array is short

Selection Sort

Sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning.[2]

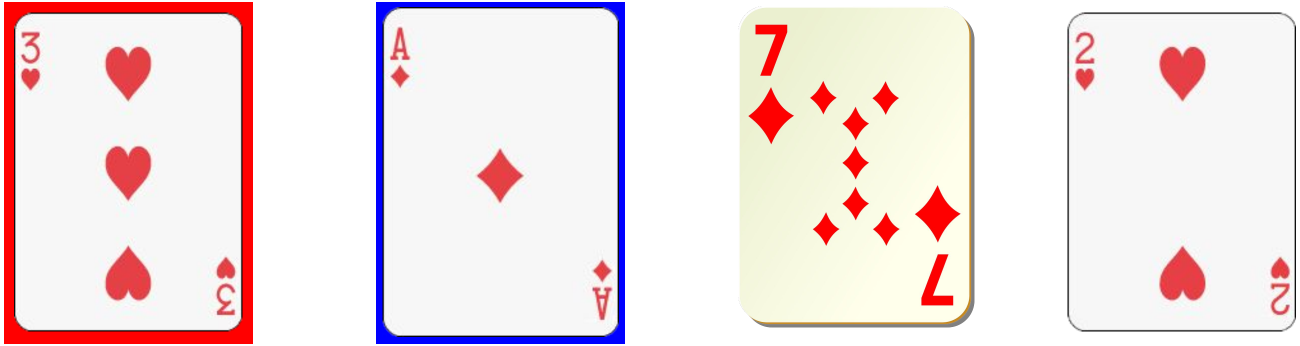
Example:
Given a set of unsorted cards:



[2] https://www.geeksforgeeks.org/selection-sort/

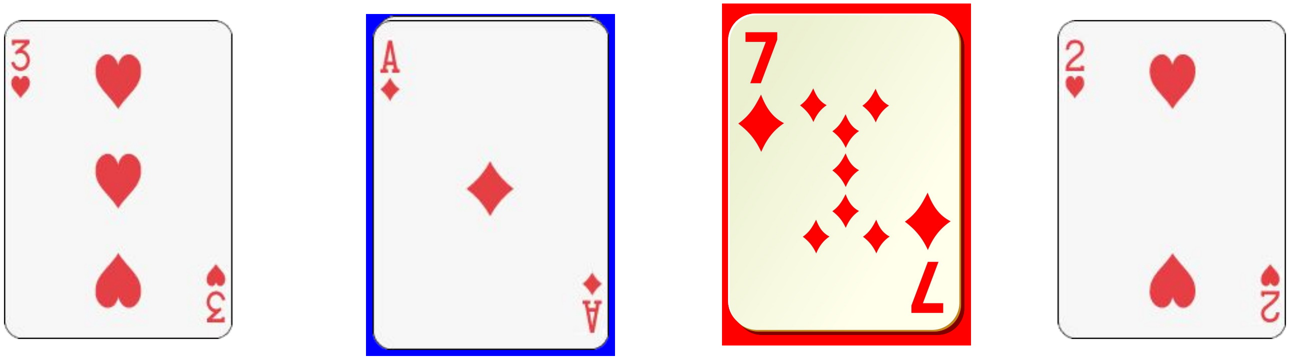
Selection Sort

It searches for the smallest item, storing it and compares it to the next one.



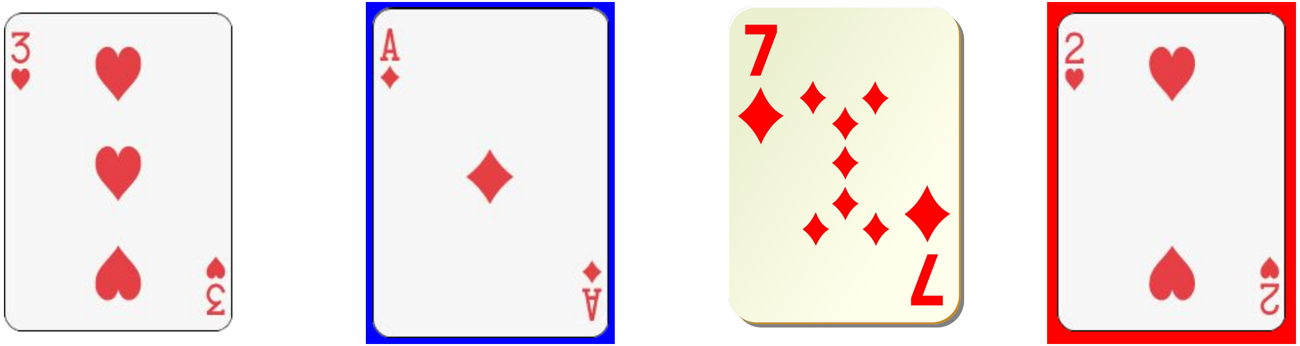
Selection Sort

It searches for the smallest item, storing it and compares it to the next one.



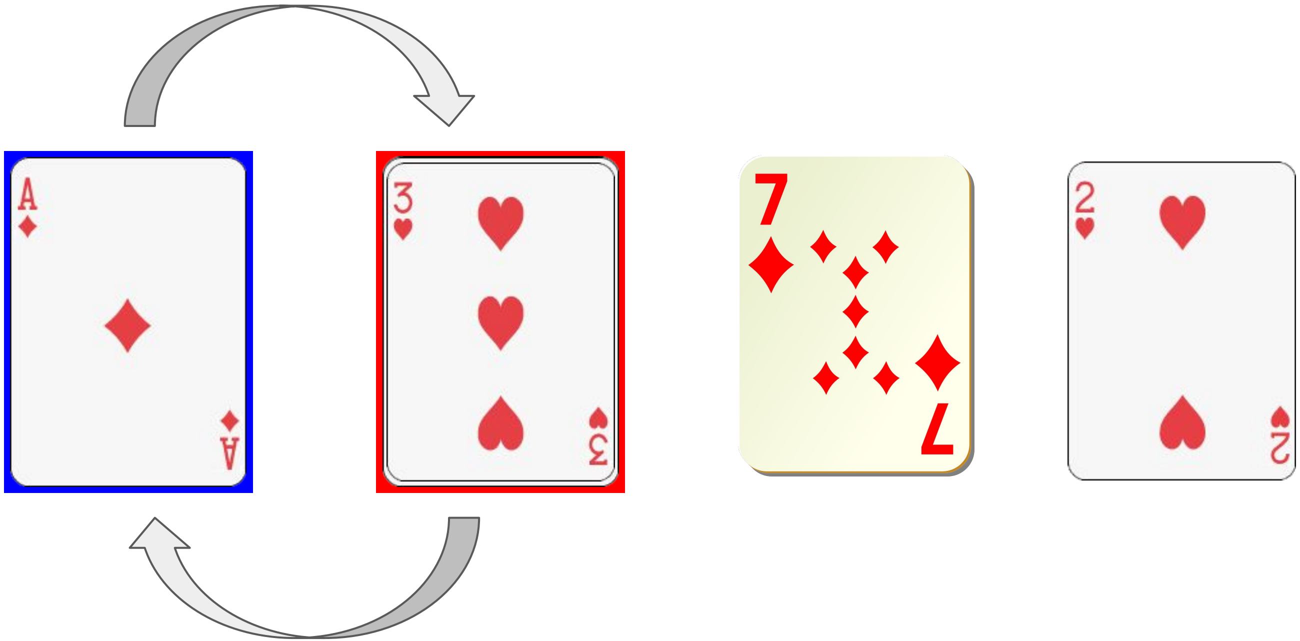
Selection Sort

Once, is compared with the whole array and has the minimum element, he exchanges. with the element where he started to compare.



In this case, 1 is the minimum elemento so the array stay the same.

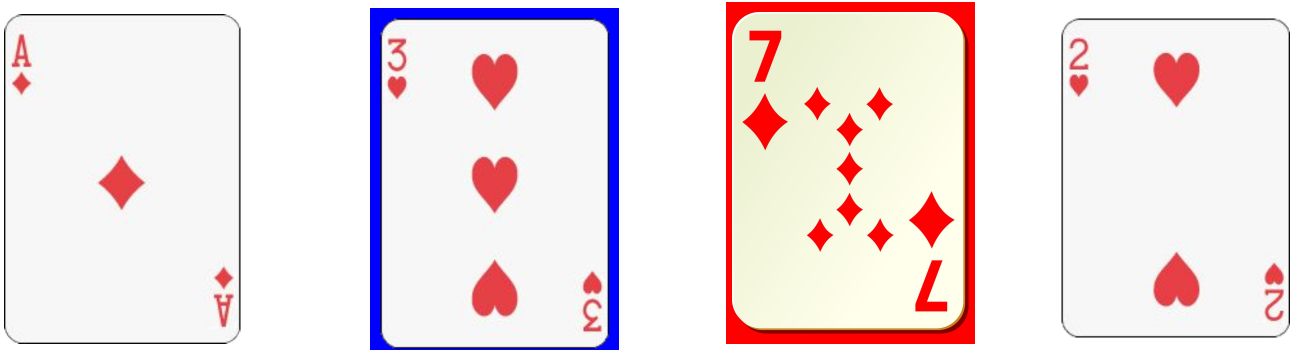
Selection Sort



In this case, 1 is the minimum elemento so the array stay the same.

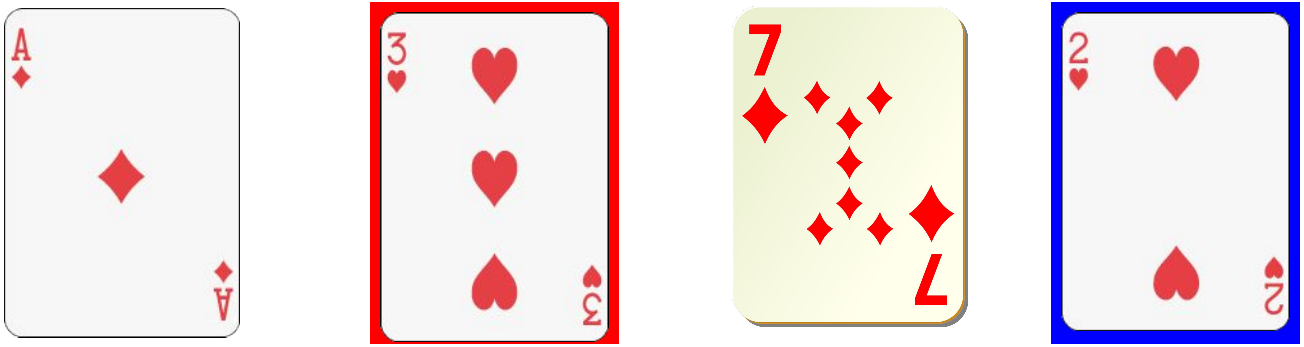
Selection Sort

Now start on the second one, store the minimum and it's compared with the next element.



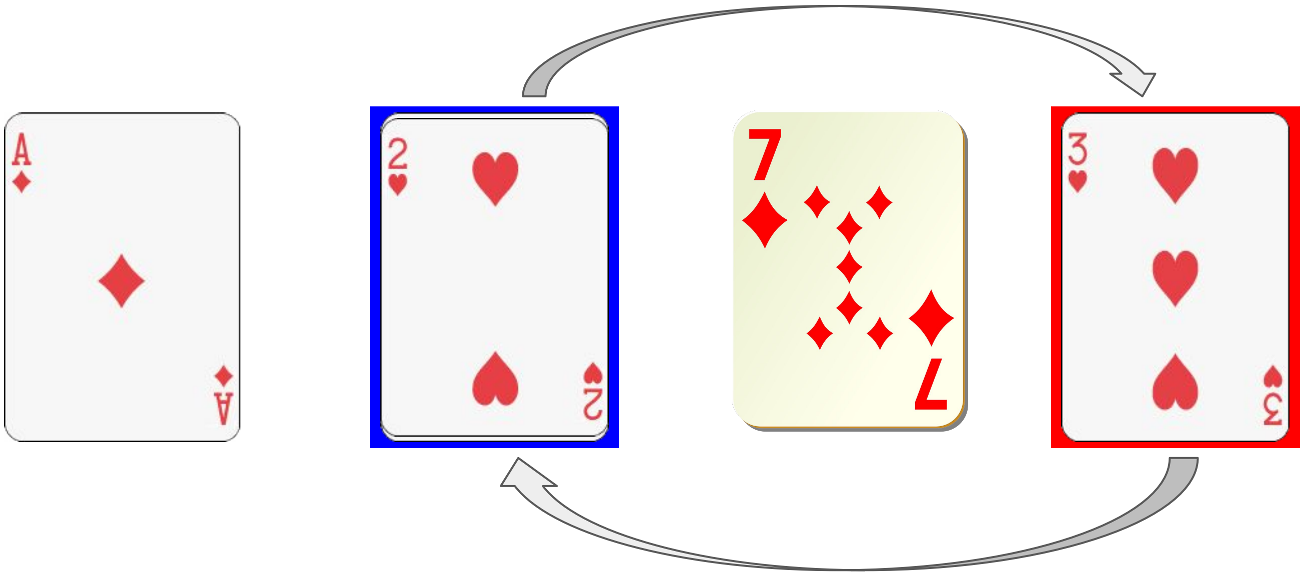
Selection Sort

Keep searching for the minimum element in the array



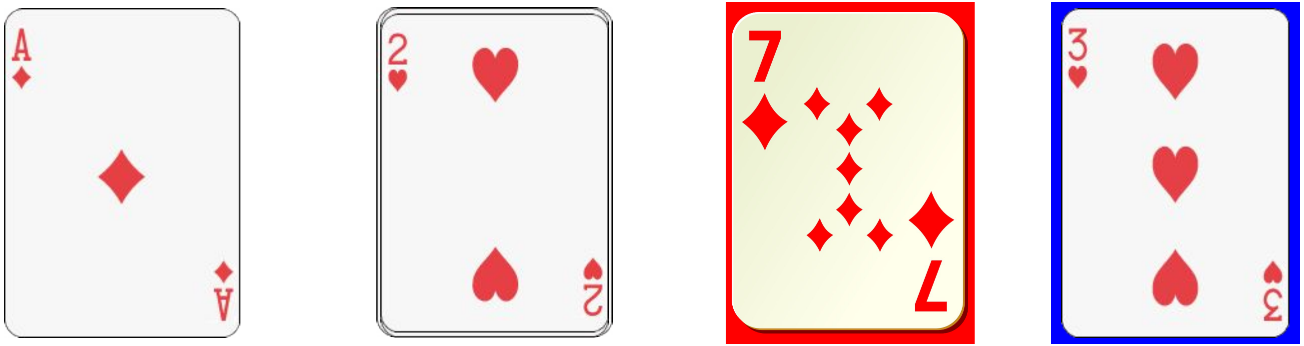
Selection Sort

Exchange.



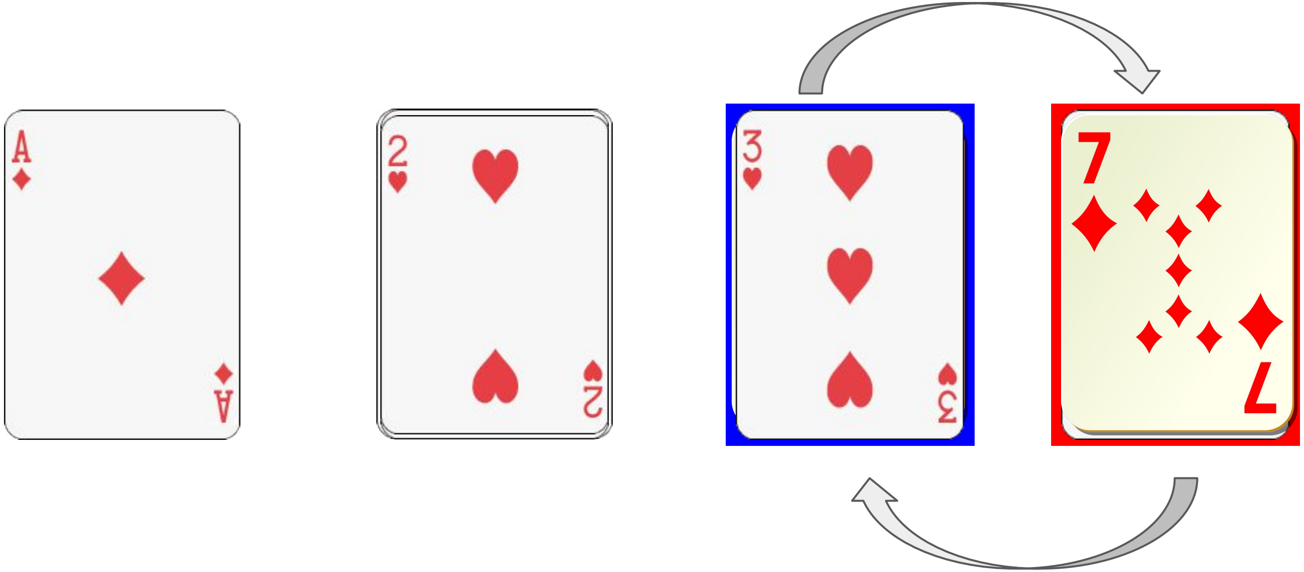
Selection Sort

Now start at the 3º position of the array, store it as a minimum element and compared it with the next position.



Selection Sort

Exchange



Selection Sort

i = Counter of the #elements of the array

j = Position to compare with the element in the position of min_idx

min_idx = Position of the minimum element

Algorithm	*B(n)*	*A(n)*	*W(n)*
SelectionSort	n^2	n^2	n^2

```c
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of unsorted subarray
    for (i = 0; i < n-1; i++)
    {
        // Find the minimum element in unsorted array
        min_idx = i;
        for (j = i+1; j < n; j++)
        if (arr[j] < arr[min_idx])
            min_idx = j;

        // Swap the found minimum element with the first element
        swap(&arr[min_idx], &arr[i]);
    }
}
```

Loop 1

Loop 2

Loop 2.1	64 25 12 22 11
Loop 2.2	11 25 12 22 64
Loop 2.3	11 12 25 22 64
Loop 2.4	11 12 22 25 64
Loop 2.5	11 12 22 25 64

Selection Sort: Summary

Computational cost:

- Time: (n^2) $$c(n) = \frac{n^2 + n}{2}$$
- Space: 1
- Stability: No

When to use it?

- When there is not enough program memory space
- When we are working with type int arrays
- When the length of the list/array is short

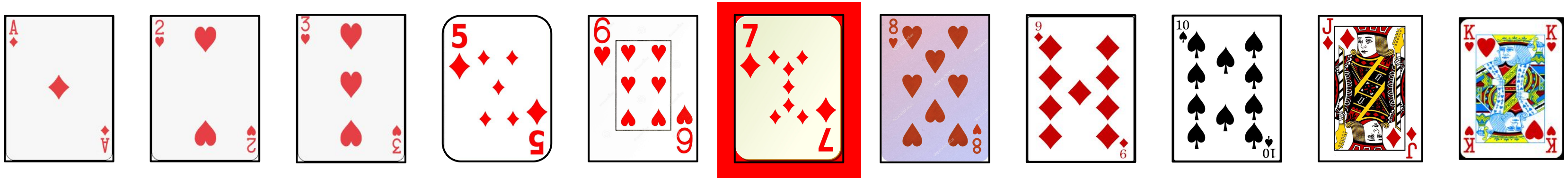
Binary Search

Search a **sorted array** by **repeatedly dividing the search interval in half**. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.[3]

[3] https://www.geeksforgeeks.org/binary-search/

Binary Search

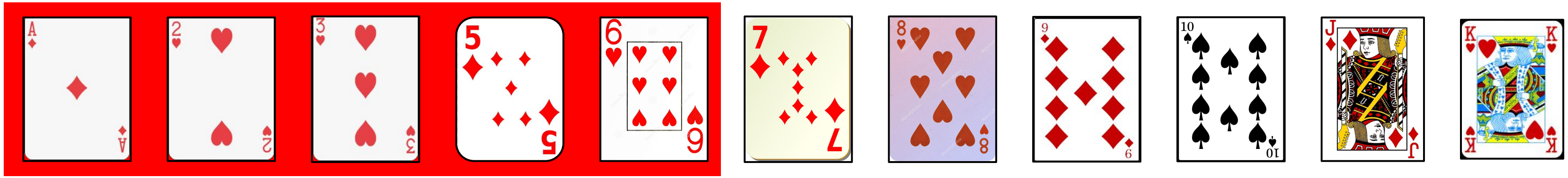
Search for the 5.



int Middle = left + (right - left)/2

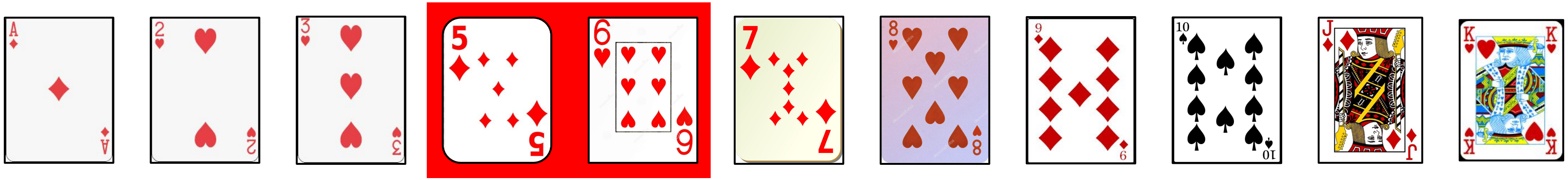
Binary Search

5 is in the group of the left.



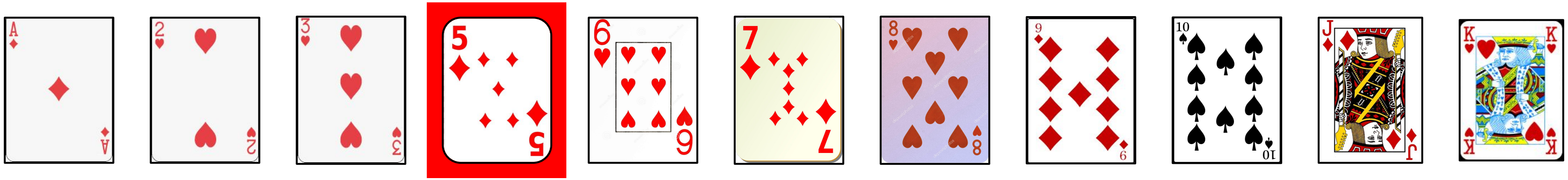
Binary Search

5 is in the group of the right.



Binary Search

The number 5 is found.



Binary Search

l = Left

R = Right

x = Valor buscado

Algorithm	*B(n)*	*A(n)*	*W(n)*
BinarySearch	1	logn	logn

Each time the loop it's repeated, the searching area is reduced in half.

```
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not
    // present in array
    return -1;
}
```

IF 1

IF 2

IF 3

Binary Search: Summary

Computational cost:

- Time: (log2(n))
- Space: 1

When to use it?

- When you want to find an element quickly
- When the array is sorted

Anexos

Algorithm	*B(n)*	*A(n)*	*W(n)*
HornerEval	n	n	n
Towers	2^n	2^n	2^n
LinearSearch	1	n	n
BinarySearch	1	logn	logn
Max, Min , MaxMin	n	n	n
InsertionSort	n	n^2	n^2
MergeSort	nlogn	nlogn	nlogn
HeapSort	nlogn	nlogn	nlogn
QuickSort	nlogn	nlogn	n^2
BubbleSort	n	n^2	n^2
SelectionSort	n^2	n^2	n^2
GnomeSort	n	n^2	n^2

Questions

1. What are the differences between Bubble sort and Selection sort?
2. What is stability?