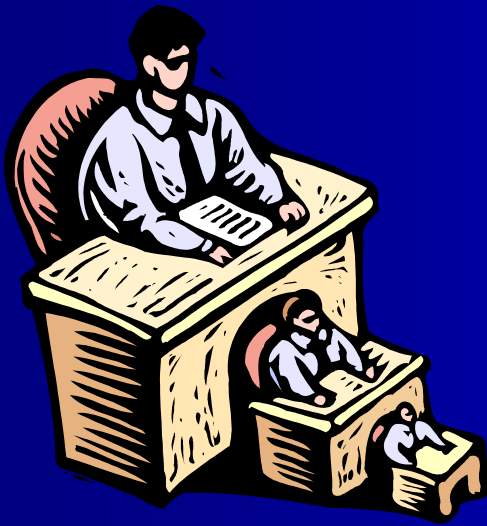




# The Selection Sort

Mr. Dave Clausen

La Cañada High School

# The Selection Sort

## Description

The Selection Sort searches (linear search) all of the elements in a list until it finds the smallest element. It “swaps” this with the first element in the list. Next it finds the smallest of the remaining elements, and “swaps” it with the second element. Repeat this process until you compare only the last two elements in the list.

# The Selection Sort Algorithm

- For each index position  $i$ 
  - Find the smallest data value in the array from positions  $i$  through  $\text{length} - 1$ , where  $\text{length}$  is the number of data values stored.
  - Exchange (swap) the smallest value with the value at position  $i$ .

# A Selection Sort Example

Smallest ?

We start by searching for the smallest element in the List.

|   |
|---|
| 6 |
| 2 |
| 1 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example

Smallest ?

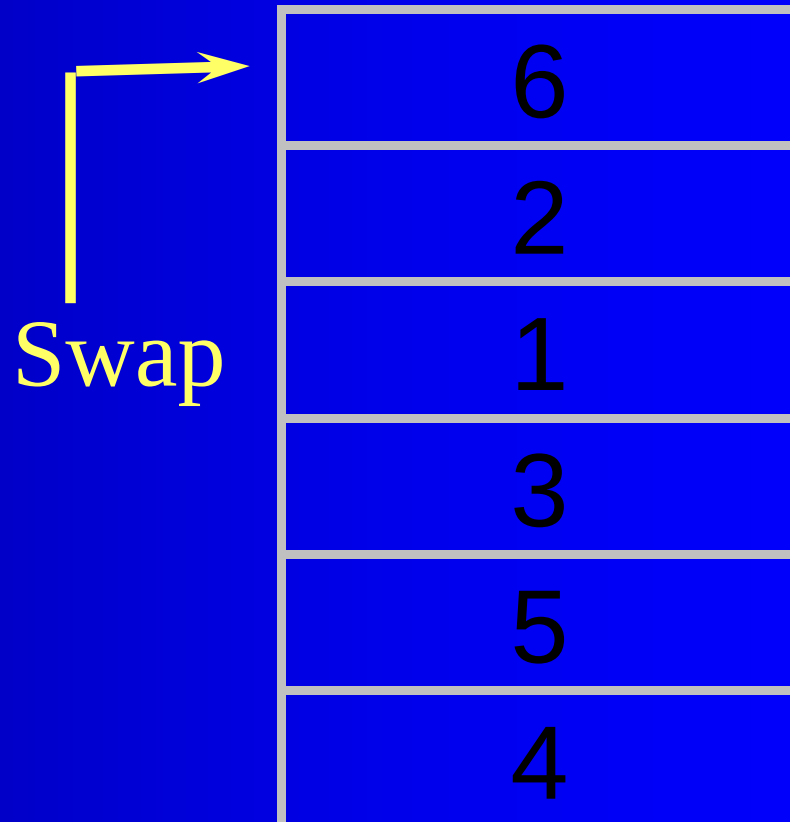
|   |
|---|
| 6 |
| 2 |
| 1 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example

Smallest !

|   |
|---|
| 6 |
| 2 |
| 1 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example



# A Selection Sort Example

Swapped

|   |
|---|
| 1 |
| 2 |
| 6 |
| 3 |
| 5 |
| 4 |



# A Selection Sort Example

Smallest ?

After the smallest element is in the first position, we continue searching with the second element and look for the next smallest element.

|   |
|---|
| 1 |
| 2 |
| 6 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example

Smallest !

In this special case, the next smallest element is in the second position already. Swapping keeps it in this position.

|   |
|---|
| 1 |
| 2 |
| 6 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example

Swapped

Swapping keeps it in this position.

|   |
|---|
| 1 |
| 2 |
| 6 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example

Smallest ?

After the next smallest element is in the second position, we continue searching with the third element and look for the next smallest element.

|   |
|---|
| 1 |
| 2 |
| 6 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example

Smallest !

|   |
|---|
| 1 |
| 2 |
| 6 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example

Swap



|   |
|---|
| 1 |
| 2 |
| 6 |
| 3 |
| 5 |
| 4 |

# A Selection Sort Example

Swapped

|   |
|---|
| 1 |
| 2 |
| 3 |
| 6 |
| 5 |
| 4 |

# A Selection Sort Example

Smallest ?

|   |
|---|
| 1 |
| 2 |
| 3 |
| 6 |
| 5 |
| 4 |



# A Selection Sort Example

Smallest ?

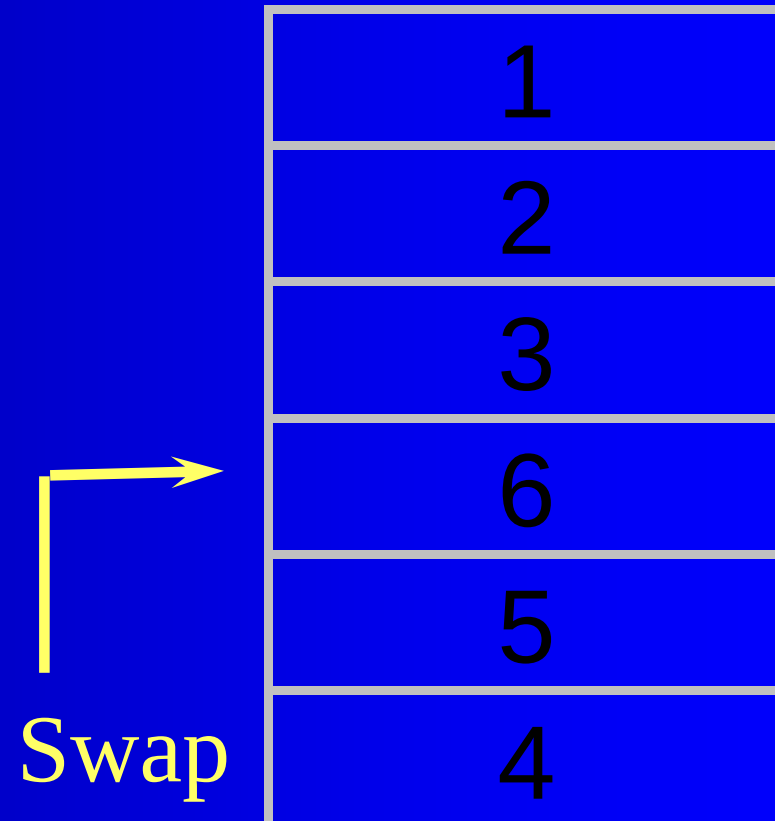
|   |
|---|
| 1 |
| 2 |
| 3 |
| 6 |
| 5 |
| 4 |

# A Selection Sort Example

Smallest !

|   |
|---|
| 1 |
| 2 |
| 3 |
| 6 |
| 5 |
| 4 |

# A Selection Sort Example



# A Selection Sort Example

Swapped

|   |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

# A Selection Sort Example

The last two elements are in order, so no swap is necessary.

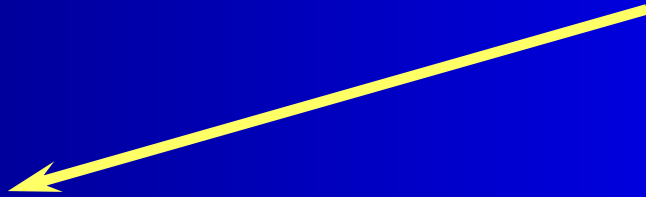
|   |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |

# What “Swapping” Means

TEMP



Place the first element into  
the Temporary Variable.



|   |
|---|
| 6 |
| 2 |
| 1 |
| 3 |
| 5 |
| 4 |

# What “Swapping” Means

TEMP

6

Replace the first element  
with the value of the  
smallest element.

|   |
|---|
| 1 |
| 2 |
| 1 |
| 3 |
| 5 |
| 4 |

# What “Swapping” Means

TEMP

|   |
|---|
| 6 |
|---|



Replace the third element  
(in this example) with the  
Temporary Variable.

|   |
|---|
| 1 |
| 2 |
| 6 |
| 3 |
| 5 |
| 4 |



# Python Source Code: Selection Sort

```
def selectionSort(listCopy):  
    i = 0  
    while i < len(listCopy)-1:  
        minIndex = i  
        j = i + 1  
        while j < len(listCopy):  
            if listCopy[j] < listCopy[minIndex]:  
                minIndex = j  
            j += 1  
        if minIndex != i:  
            swap(listCopy, minIndex, i)  
        i += 1
```

# Python Code for Swap Procedure

```
def swap(listCopy, x, y):  
    temp = listCopy [x]  
    listCopy [x] = listCopy [y]  
    listCopy [y] = temp
```

# Java Source Code: Selection Sort

```
public static void selectionSort(int[] list)
{
    for (int index = 0; index < list.length - 1; index++)
    {
        int minIndex = findMinimum(list, index );
        if (minIndex != index )
            swap(list, index , minIndex);
    }
}
```

# Java Code For Find Minimum

```
public static int findMinimum(int[] list, int first)
{
    int minIndex = first;

    for (int index = first + 1; index < list.length; index++)
        if (list[index] < list[minIndex])
            minIndex = index ;

    return minIndex;
}
```

# Java Code for Swap Procedure

```
public static void swap(int[] list, int x, int y)
{
    int temp = list[x];
    list[x] = list[y];
    list[y] = temp;
}
```

# C ++ Code For Selection Sort

```
void Selection_Sort (apvector <int> & v)
{
    int min_index = 0;
    for (int index = 0; index < v.length( ) - 1; ++index)
    {
        min_index = Find_Minimum (v, index);
        if (min_index != index)
            Swap_Data ( v [index], v [min_index])
    }
}

// Selection_Sort
```

# C ++ Code For Find Minimum

```
int Find_Minimum (const apvector <int> & v, int first)
{
    int min_index = first;
    for (int index = first + 1; index < v.length( ); ++index)
        if ( v[index] < v[min_index])
            min_index = index;
    return min_index;
}
// Find_Minimum
```

# C ++ Code for Swap Procedure

```
void Swap_Data (int &number1, int &number2)
{
    int temp;
    temp = number1;
    number1 = number2;
    number2 = temp;
}
// End of Swap_Data function
```



# Pascal Code For Selection Sort

```
procedure SelectionSort (var IntArray: IntNumbers);  
  var  element, SmallIndex, index: integer;  
begin  
  for element := 1 to (MaxNum - 1) do  
    begin  
      SmallIndex := element;  
      for index := (element + 1) to MaxNum do  
        if IntArray [index] < IntArray [SmallIndex]  
          then SmallIndex := index;  
      Swap (IntArray [element], IntArray [SmallIndex])  
    end;  
  end;  
end;  {SelectionSort}
```

# Pascal Code for Swap Procedure

```
procedure Swap (var number1, number2: integer);  
var  
    temp: integer;  
begin  
    temp := number1;  
    number1 := number2;  
    number2 := temp  
end;    {Swap}
```

# BASIC Code For Selection Sort

```
8000 REM #####
8010 REM Selection Sort
8020 REM #####
8030 FOR ELEMENT = 1 TO MAXNUM - 1
8040 ::SmallIndex = element
8050 ::FOR INDEX = (element + 1) TO MAXNUM
8060 ::::::IF N (INDEX) < N (SmallIndex) THEN SmallIndex = INDEX
8070 ::NEXT INDEX
8080 ::TEMP = N (element)
8090 ::N (element) = N (SmallIndex)
8100 ::N (SmallIndex) = TEMP
8110 NEXT ELEMENT
8120 RETURN
```

# Big - O Notation

Big - O notation is used to describe the efficiency of a search or sort. The actual time necessary to complete the sort varies according to the speed of your system. Big - O notation is an approximate mathematical formula to determine how many operations are necessary to perform the search or sort. The Big - O notation for the Selection Sort is  $O(n^2)$ , because it takes approximately  $n^2$  passes to sort the elements.