Sorting Algorithms

Topics

- Sorting Algorithms
- Bubble Sort
- Swapping Items
- Comparing Strings
- Merge Sort

Sorting

Sorting Algorithms

- **Sorting** puts elements of a list in a certain order.
- Most common orderings are **numerical** or **alphabetical**.

Why do we sort?

- Essential for **searching** and **merging.** The list must be sorted before it can be searched or merged efficiently.
- For **UIs** and user experience (ex: sort list by price low-high). Humans like to see lists arranged in ways that help them.
- For **grouping** and **decision making** (ex: Z-order)

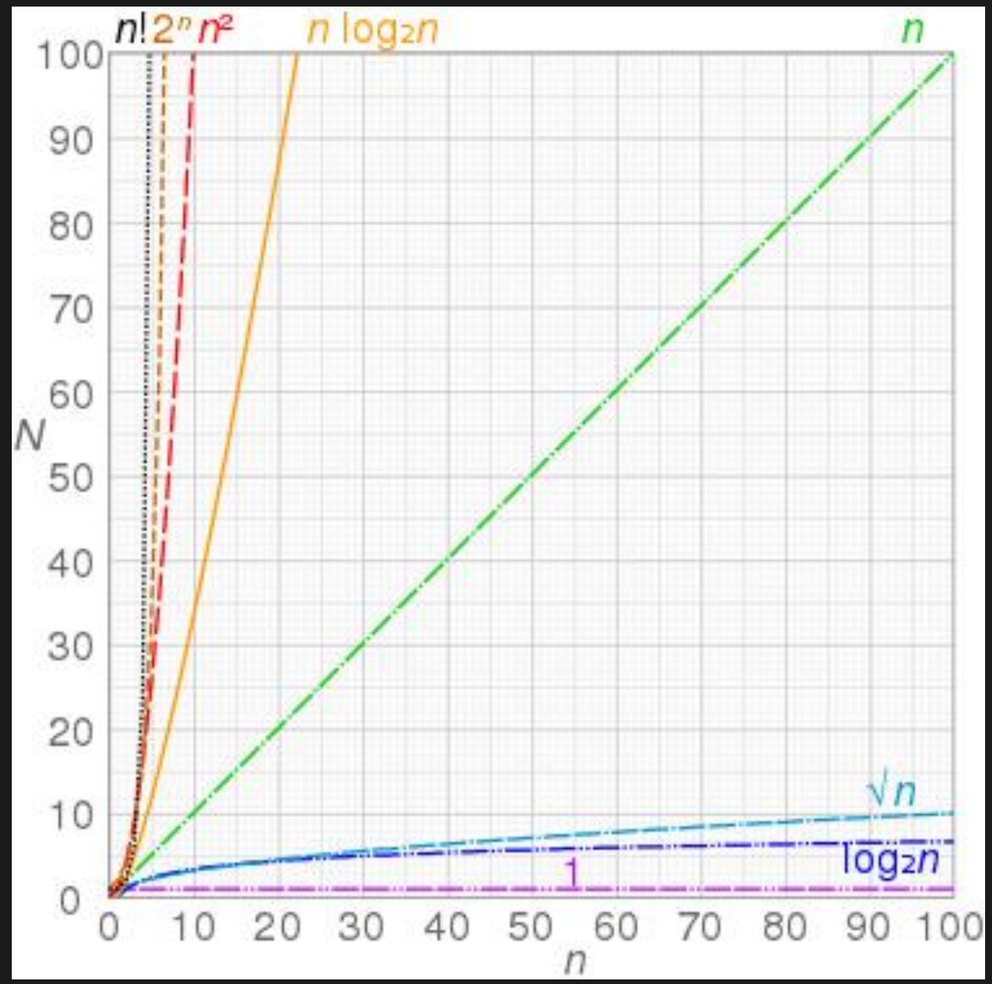
Performance

Computational Complexity

Best

O(1)	constant
O(log n)	logarithmic
O(n)	linear
O(n log n)	loglinear
$O(n^2)$	quadratic
$O(2^n)$	exponential
O(n!)	factorial

Worst



Sorting Algorithms

ALGORITHM	WORST	BEST	AVG
Heap Sort	O(n log n)	O(n log n)	O(n log n)
Quick Sort	$O(n^2)$	O(n log n)	O(n log n)
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$
Bubble Sort	$O(n^2)$ comparisons, $O(n^2)$ swaps	O(n) comps, O(1) swaps	$O(n^2)$ comparisons, $O(n^2)$ swaps

Bubble Sort

Bubble Sort

- The Algorithm
  1. Compare each pair of adjacent elements from the beginning of the array
     - If they are in reversed order, swap them
  2. If at least one swap was made, repeat step 1

```
procedure bubbleSort(A : list of sortable items)
    n := length(A)
    repeat    //this is the start of a loop
        swapped := false
        for i := 1 to n - 1 inclusive do
            if A[i - 1] > A[i] then
                swap(A[i - 1], A[i]) //you'll need swap code here
                swapped = true
            end if
        end for
        n := n - 1
    while we swapped something //this is the end of the loop
end procedure
```

Swapping Items

How Do We Swap?

EXAMPLE: swap indexes 10 and 11

int k = 10, j = 11; //indexes

1) Store **j** item in a temporary variable

```
int temp = numbers[ j ];
```

2) Copy the k item to j location

```
numbers[ j ] = numbers[ k ];
```

3) Copy the temporary variable to the k location

```
numbers[ k ] = temp;
```

Swap Method Challenge

1. Write a Swap method that will swap the items in an array at the specified indexes.
    1. Parameters to the method:
        1. The int array
        2. The first index
        3. The second index
2. Call Swap from Main.
3. Print the array.

```
int k = 10, j = 11; //indexes

int temp = numbers[ j ];
numbers[ j ] = numbers[ k ];
numbers[ k ] = temp;
```

LINKS

VIDEOS

Comparing Strings

String Comparison

- How do we compare strings?
- Equal is easy – the string class already handles equality for us.
  - Use the == operator or the Equals method
- But what about < or >?
  How do we know that one string is less than another for sorting purposes?
- Use the **CompareTo** method on string.

String Comparison

- **string1.CompareTo(string2);**
  - Returns **0** if **equal**
  - Returns **-1** if string1 **<** string2
  - Returns **1** if string1 **>** string2

- EXAMPLE:
  string str1 = “A”;
  string str2 = “B”;
- str1.CompareTo(str2) returns -1 meaning “A” is < “B”
- str2.CompareTo(str1) returns 1 meaning “B” is > “A”

Compare 2 strings Challenge

1. Create a method called CompareStrings
2. Use ReadLine to get 2 strings from the user.
3. Compare the strings using CompareTo.
4. Print a message about the strings. LESS than, GREATER than, or EQUAL to.
5. Call CompareStrings from Main

```
string s1 = "Batman", s2 = "Aquaman";
int compResult = s1.CompareTo(s2);
```

LINKS

CompareTo

VIDEOS

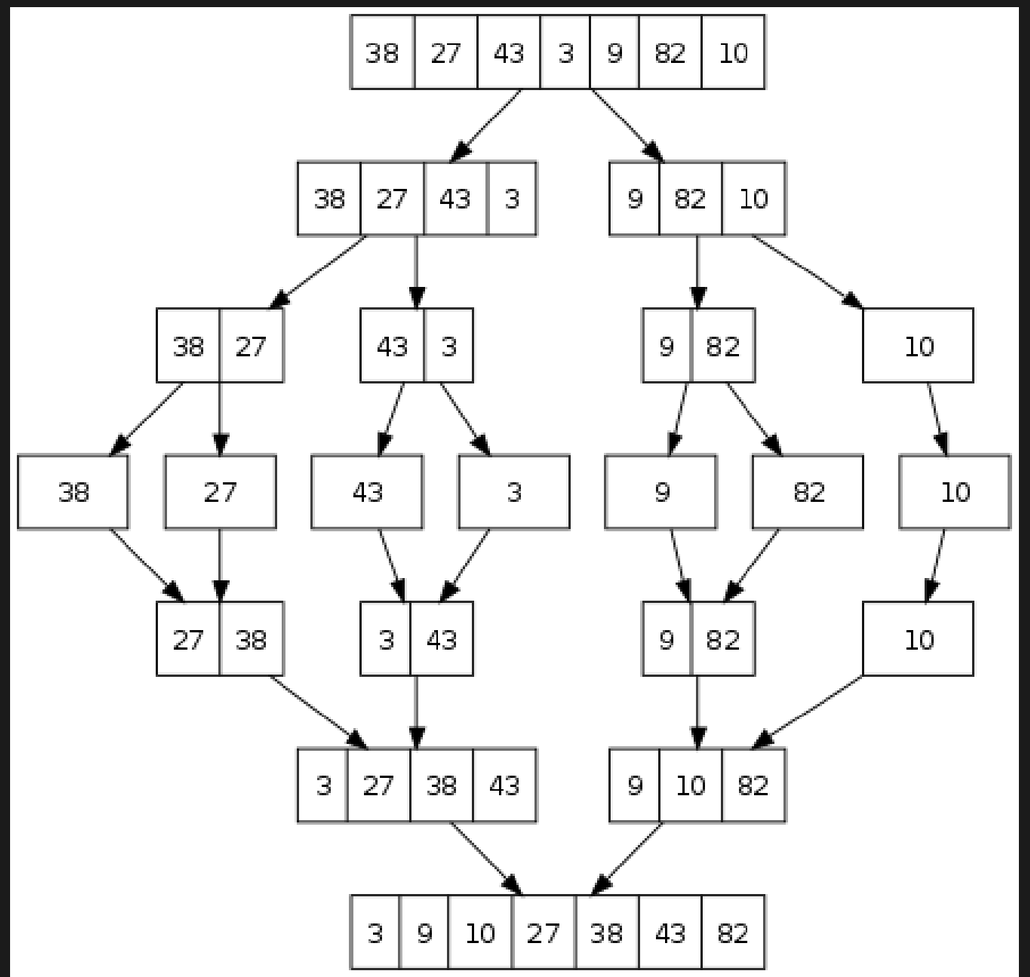
Merge Sort

Merge Sort

- The Algorithm
  1. Divide the list into n sublists that are all 1 element long.
  2. Merge the sublists to make new ***sorted*** lists.
  3. Continue merging until only 1 list remains.

Merge Sort

- https://en.wikipedia.org/wiki/File:Merge_sort_algorithm_diagram.svg



Merge Sort: Split Step

1) Split pseudocode

```
function merge_sort(list m) is

    // exit condition. A list of zero or one elements is sorted, by definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide the list into sublists
    // consisting of the first half and second half of the list. This assumes lists start at index 0.

    var left := empty list
    var right := empty list

    for each x with index i in m do
        if i < (length of m)/2 then
            add x to left
        else
            add x to right

    // Recursively sort both sublists.
    left := merge_sort(left)
    right := merge_sort(right)

    // Then merge the now-sorted sublists.
    return merge(left, right)
```

Merge Sort: Merge Step

2) Merge pseudocode

```
function merge(left, right) is
    var result := empty list

    while left is not empty and right is not empty do

        if first(left) ≤ first(right) then
            add first(left) to result
            left := rest(left) //remove first item from left
        else
            add first(right) to result
            right := rest(right) //remove first item from right

    // Either left or right may have elements left; consume them.
    // (Only one of the following loops will actually be entered.)

    while left is not empty do
        add first(left) to result
        left := rest(left)

    while right is not empty do
        add first(right) to result
        right := rest(right)

    return result
```

Split a List Challenge

1. Create a method called Split that takes one list of ints as a parameters.
2. In the method, split the list into 2 halves: left and right.
3. Print the left half then print the right half. Before printing each half, print a title for the list: “Left Half” and “Right Half”
4. Call Split from Main.

```
    var left := empty list
    var right := empty list

for each x with index i in m do
    if i < (length of m)/2 then
        add x to left
    else
        add x to right
```

LINKS

VIDEOS