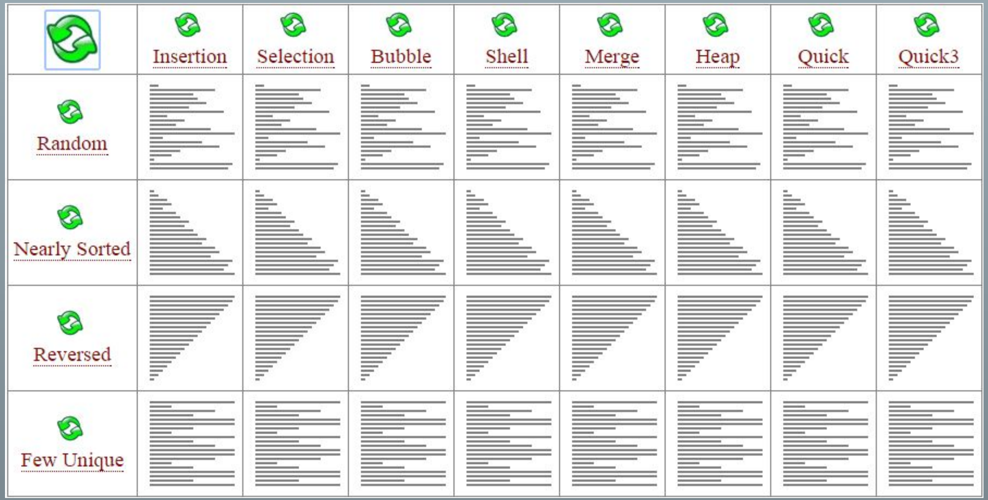
A Deep Dive On Sorting

By: Adan H.

Sorting: What is it?

Sorting is the process of just taking a scrambled set of data and unscrambling it to fit whatever use case that data is required for.

The idea is simple but to put it into use, we need to have an algorithm or algorithms that fit the scenario to maximize speed and efficiency.



What's to talk about?

There's tons of things to talk about, there's the beginning of sorting as well as the complexity of sorting algorithms increasing as the years go by. I'll only be able to talk about the surface level but that's still a lot to cover.

Things that are nice to know

The beginning of sorting algorithms can be traced back all the way to the early 50's during the beginning of computers. Sorting in the early days were viable for smaller datasets but got less viable as the datasets grew, that lead people to figure out more effective algorithms to deal with the increasing data.

Importance of Sorting:

You wouldn't want a program to just sit there hopelessly trying to sift through a 20k+ dataset for a commercial application. Sorting algorithms help speed things up and just make life quick and easy.

What are algorithms in the first place?

Algorithms would be the process that’s used to solve a problem or to perform a computation.

They follow a set of rules, in our case being computer code, to perform these tasks.

What kinds of sorting algorithms are there?

There are many kinds of algorithms that fit different requirements, those requirements being:

- Memory usage
- Overall stability
- The method that they use to sort themselves

The methods that they use also is a topic itself but some include:

- Sequential sorting
- Parallel sorting

Now I’ll move onto a deeper dive on more of the simpler algorithms

A Deeper Dive on Bubble Sort:

Earliest Known Description/Usage:

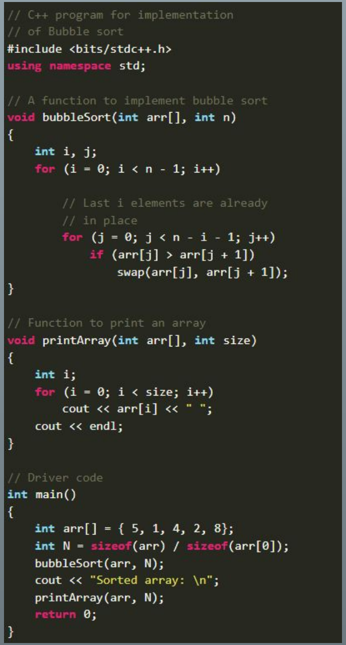
A 1956 paper by Edward Harry Friend called *Sorting on electronic computer systems*.

Complexity number:

O (n^2)

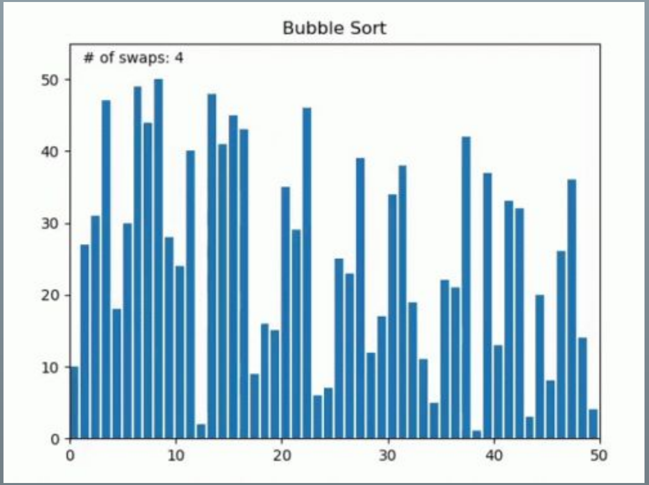
A good start to sorting but can easily be improved upon

C++ Example



```cpp
// C++ program for implementation
// of Bubble sort
#include <bits/stdc++.h>
using namespace std;

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}

// Function to print an array
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = { 5, 1, 4, 2, 8};
    int N = sizeof(arr) / sizeof(arr[0]);
    bubbleSort(arr, N);
    cout << "Sorted array: \n";
    printArray(arr, N);
    return 0;
}
```

Visual Example



A Deeper Dive on Selection Sort:

Earliest Known Usage/Description:

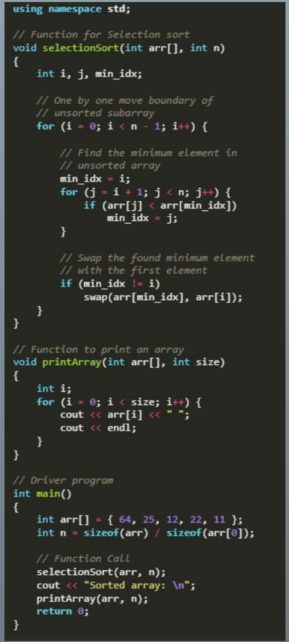
I couldn’t manage to find a specified year or time for it, but it’s agreed that selection sorting was also one of the earlier sorting algorithms.

Complexity Number:

O (n^2)

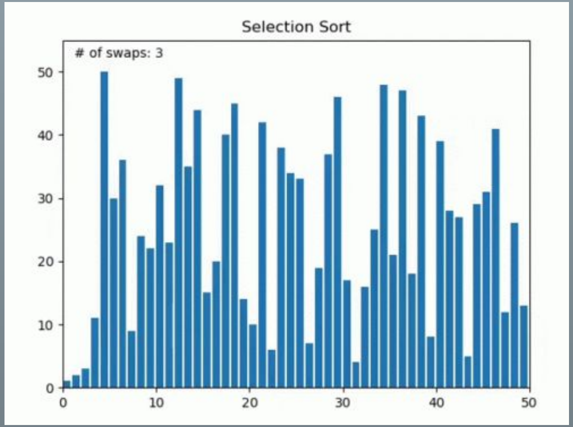
Even with the same complexity number, the visual shows how much faster it is compared to bubble sort.

C++ Example



```
using namespace std;

// Function for Selection sort
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
        if (min_idx != i)
            swap(arr[min_idx], arr[i]);
    }
}

// Function to print an array
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++) {
        cout << arr[i] << " ";
        cout << endl;
    }
}

// Driver program
int main()
{
    int arr[] = { 64, 25, 12, 22, 11 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // Function Call
    selectionSort(arr, n);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Visual Example



A Deeper Dive on Insertion Sort:

Earliest Known Usage/Description:

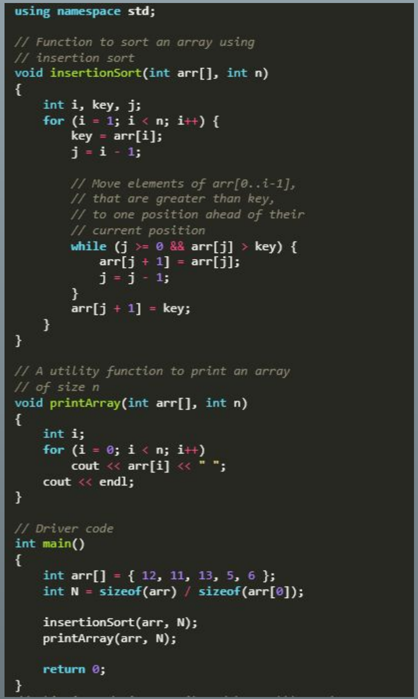
The earliest known mention comes from before computers and can be found from Konrad Zuse and was still use was still for computing.

Complexity Number:

O (n^2)

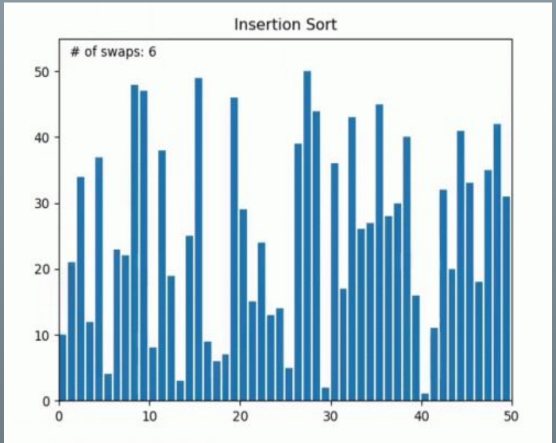
Even with the same complexity number, the visual shows how much faster it is compared to bubble sort.

C++ Example



```cpp
using namespace std;

// Function to sort an array using
// insertion sort
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        // Move elements of arr[0..i-1],
        // that are greater than key,
        // to one position ahead of their
        // current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

// A utility function to print an array
// of size n
void printArray(int arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int N = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, N);
    printArray(arr, N);

    return 0;
}
```

Visual Example



A Deeper Dive on Shell Sort:

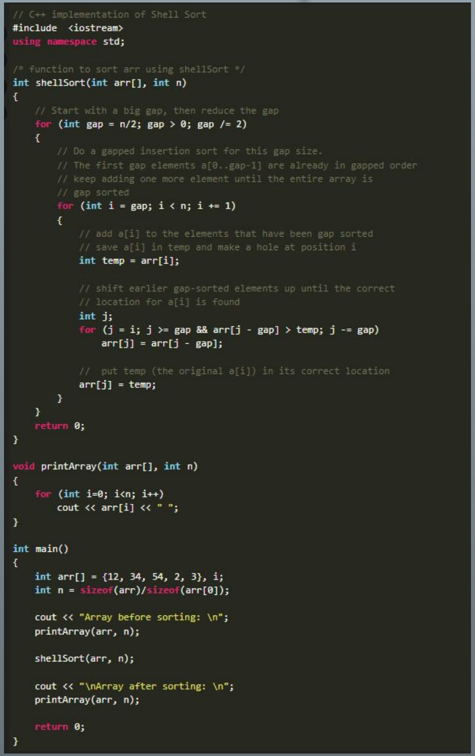
Earliest known Description/Usage:

Donald Shell published the first version of shell sort back in 1959 and is a variation of insertion sort.

Complexity number:

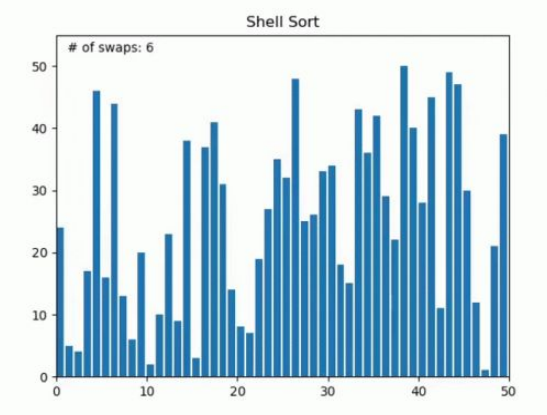
O (n^2)

C++ Example



```
// C++ implementation of Shell Sort
#include  <iostream>
using namespace std;

/* function to sort arr using shellSort */
int shellSort(int arr[], int n)
{
    // Start with a big gap, then reduce the gap
    for (int gap = n/2; gap > 0; gap /= 2)
    {
        // Do a gapped insertion sort for this gap size.
        // The first gap elements a[0..gap-1] are already in gapped order
        // keep adding one more element until the entire array is
        // gap sorted
        for (int i = gap; i < n; i += 1)
        {
            // add a[i] to the elements that have been gap sorted
            // save a[i] in temp and make a hole at position i
            int temp = arr[i];

            // shift earlier gap-sorted elements up until the correct
            // location for a[i] is found
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];

            //  put temp (the original a[i]) in its correct location
            arr[j] = temp;
        }
    }
    return 0;
}

void printArray(int arr[], int n)
{
    for (int i=0; i<n; i++)
        cout << arr[i] << " ";
}

int main()
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Array before sorting: \n";
    printArray(arr, n);

    shellSort(arr, n);

    cout << "\nArray after sorting: \n";
    printArray(arr, n);

    return 0;
}
```

Visual Example



A Deeper Dive on Tim Sort:

Earliest known Description/Usage:

Tim sort came about in 2002 by Tim Peters originally for the Python programming language.

Complexity number:

O(n)

Link to code: https://www.geeksforgeeks.org/shellsort/

Link to audio/visual example: https://www.youtube.com/watch?v=NVIjHj-IrT4

A Deeper Dive on Bogo Sort:

Earliest Known Description/Usage:

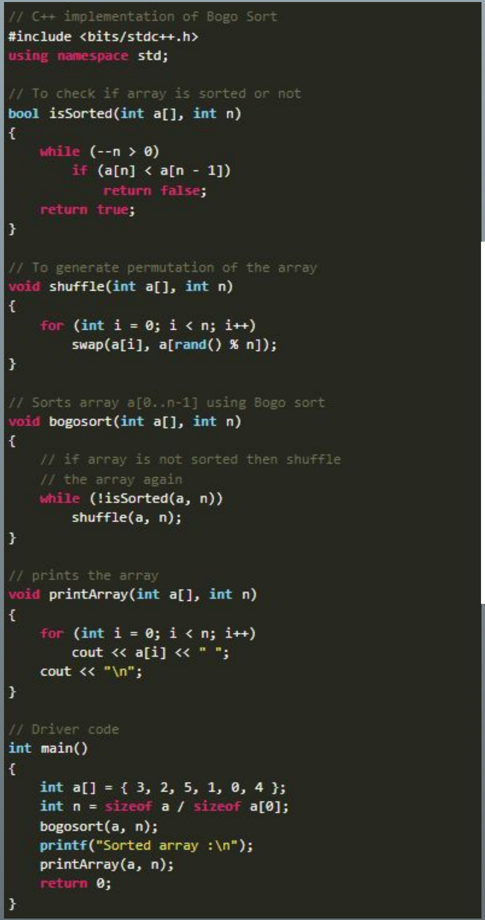
Bogo sort was originally mentioned as a part of a paper called *Pessimal algorithms and simplexity analysis* by Andrei Broder and Jorge Stolfi. It was never meant to be used in a practical setting.

Complexity number:

O (?)

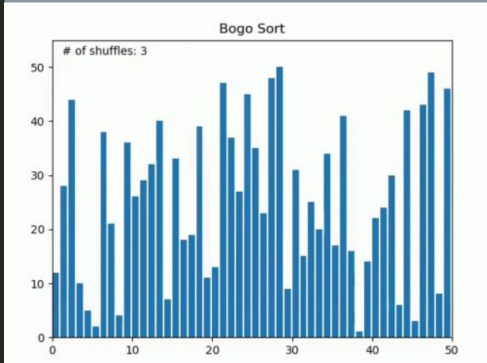
There’s no upper bound on this algorithm and is incredibly inefficient.

C++ Example



```
// C++ implementation of Bogo Sort
#include <bits/stdc++.h>
using namespace std;

// To check if array is sorted or not
bool isSorted(int a[], int n)
{
    while (--n > 0)
        if (a[n] < a[n - 1])
            return false;
    return true;
}

// To generate permutation of the array
void shuffle(int a[], int n)
{
    for (int i = 0; i < n; i++)
        swap(a[i], a[rand() % n]);
}

// Sorts array a[0..n-1] using Bogo sort
void bogosort(int a[], int n)
{
    // if array is not sorted then shuffle
    // the array again
    while (!isSorted(a, n))
        shuffle(a, n);
}

// prints the array
void printArray(int a[], int n)
{
    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << "\n";
}

// Driver code
int main()
{
    int a[] = { 3, 2, 5, 1, 0, 4 };
    int n = sizeof a / sizeof a[0];
    bogosort(a, n);
    printf("Sorted array :\n");
    printArray(a, n);
    return 0;
}
```

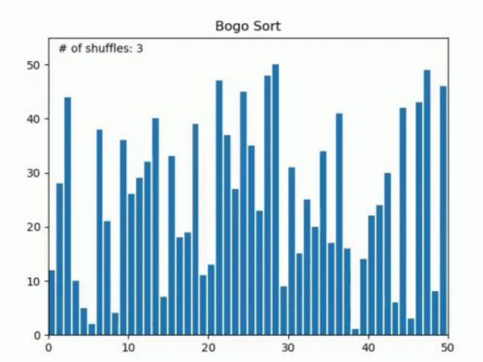
Visual example:
This will never finish



Conclusion:

Almost every sorting algorithm has their place, from small projects to massive commercial uses that need to be quick enough for millions of people to use simultaneously. Algorithms will only get more and more efficient to go alongside hardware advancements, this is clear.

Still going…


Bogo Sort
# of shuffles: 3