Week 9 - ADA

PROJECT MILESTONE

Jocelin | Nicholas | Tiffany

PREVIOUS TOPIC

Comparison between

Greedy, Divide & Conquer and

Dynamic Programming

WEIRD SORTING ALGORITHMS

Spaghetti

Identifying and then removing the longest spaghetti rod

Stalin

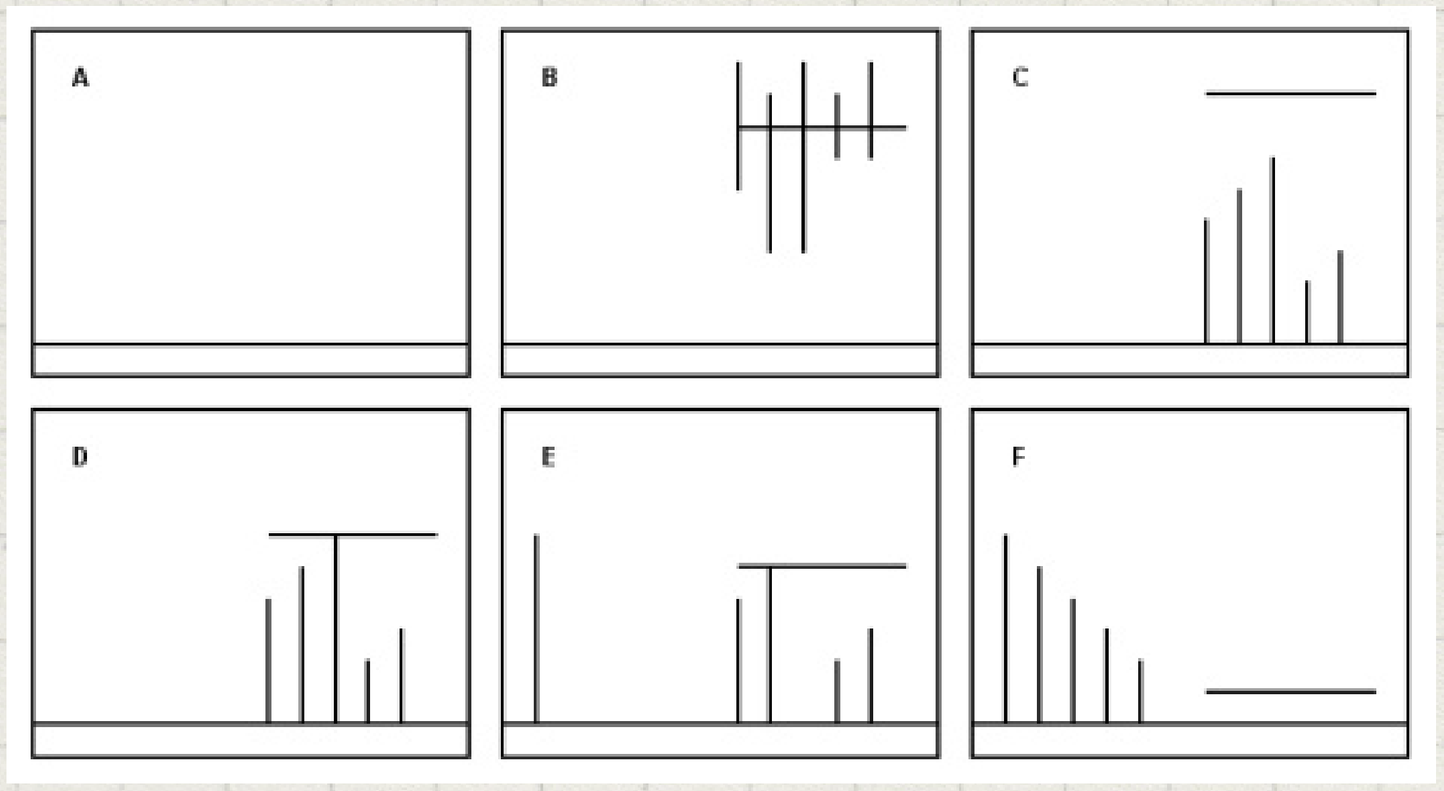
Eliminating numbers that are not positioned correctly.

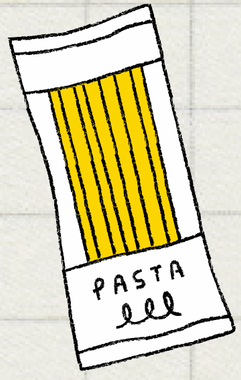
Sleep

The element having the least amount of sleeping time wakes up first and the number gets printed

Cocktail Shaker

A variation of bubble sort. raverses back and forth identifying and sort the smallest and largest, second smallest and second largest, and so on





SPAGHETTI VISUALIZATION

STALIN SORT PSEUDO-CODE

```
FUNCTION STALINSORT(A : LIST OF SORTABLE ITEMS)
    N := LENGTH(A)
    BIGGER := 0
    B SET EMPTY LIST

    FOR I := 0 TO N NOT INCLUSIVE DO
        IF A[I] >= BIGGER THEN
          BIGGER := A[I]
          B.PUSH(A[I])
        END IF
    END FOR

    RETURN B
END FUNCTION
```

SLEEP SORT PSEUDO-CODE

```
FOR N IN ARRAY
CREATE A NEW THREAD
MAKE IT SLEEP FOR ARRAY[N]
MILLISECONDS
PRINT ARRAY[N]
END
WAIT FOR ALL PROCESSES TO
FINISH
```

COCKTAIL SHAKER
PSEUDO-CODE

Algorithm 1: Cocktail sort

```
Data: Input array A[]
Result: Sorted A[]
int i,swapped;
do
    swapped=0;
    foreach i in 0 to length(A)-2 do
        if A[i] > A[i + 1] then
            swap(A[i],A[i+1]);
            swapped=1;
        end
    end
    if !swapped then
        break do-while loop;
    end
    swapped=0;
    foreach i in length(A)-2 to 0 do
        if A[i] > A[i + 1] then
            swap(A[i],A[i+1]);
            swapped=1;
        end
    end
while swapped;
```

CODE DEMO TIME!

STALIN SORT

```cpp
#include <vector>
#include <iostream>
#define KEY StalinSort
#define minor main
#define pointer *
#define getPointer &
#define print std::cout
#define array std::vector
#define add push_back

//This implementation of stalin sort is easier to read and understand
template <typename T>
array<T> stalinSort(T pointer arr, size_t count)
{
    array<T> orderedData;
        orderedData.add(arr[0]);

        unsigned int last = 0;
        for(unsigned int i = 1;i < count;i++)
    {
        if (arr[i] >=  orderedData[last])
        {
            last ++;
                orderedData.add(arr[i]) ;
        }
    }
    return orderedData ;
}

int minor()
{
    array<int> KEY ;
        KEY.add(1) ;
        KEY.add(3) ;
        KEY.add(2) ;
        KEY.add(5) ;
        KEY.add(3) ;
        KEY.add(8) ;
        KEY.add(7) ;
        KEY.add(9) ;

        array<int> ordered = stalinSort<int>(getPointer KEY[0], KEY.size());

    for (unsigned int i = 0 ; i < ordered.size() ; i ++)
        print << ordered[i] << "\n";

    return 0 ;
}
```

```
Microsoft Visual Studio Debug Console
Sorted array:  1  3  5  8  9
```

SORTED!

MERCIFUL DICTATOR

```cpp
#include<iostream>
#include<vector>
using namespace std;

// Function to sort the array
void variationStalinsort(vector<int> arr)
{
    int j = 0;

    while (true)
    {
        int moved = 0;

        for (int i = 0;
            i < (arr.size() - 1 - j); i++)
        {
            if (arr[i] > arr[i + 1])
            {
                vector<int>::iterator index;
                int temp;
                index = arr.begin() + i + 1;
                temp = arr[i + 1];
                arr.erase(index);
                arr.insert(arr.begin() + moved, temp);
                moved++;
            }
        }

        j++;

        if (moved == 0)
        {
            break;
        }
    }
    cout << "Sorted array: ";
    for (int i = 0; i < arr.size(); i++)
    {
        cout << arr[i] << " ";
    }
}

// Driver Code
int main()
{
    vector<int> arr = { 1, 3, 2, 5, 3, 8, 7, 9 };

    // Function call
    variationStalinsort(arr);
}
```

SORTED!

Microsoft Visual Studio Debug Console

```
Sorted array: 1 2 3 3 5 7 8 9
```

COCKTAIL SORT

```cpp
#include <iostream>

using namespace std;

void CocktailSort(int a[], int n)
{
    bool swapped = true;
    int start = 0;
    int end = n - 1;

    while (swapped) {
        //set default swapped variable to false
        swapped = false;

        // loop from left to right same as
        for (int i = start; i < end; ++i) {
            if (a[i] > a[i + 1]) {
                swap(a[i], a[i + 1]);
                swapped = true;
            }
        }

        // if nothing gets swapped then break the loop cz its basically sorted
        if (!swapped)
            break;
        // Repeat with end point moved back by 1
        swapped = false;


        --end;

        for (int i = end - 1; i >= start; --i) {
            if (a[i] > a[i + 1]) {
                swap(a[i], a[i + 1]);
                swapped = true;
            }
        }

        // increase the starting point, because
        // the last stage would have moved the next
        // smallest number to its rightful spot.
        ++start;
    }
}

/* Prints the array */
void printArray(int a[], int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}

// vroom vroom
int main()
{
    int a[] = { 1, 3, 2, 5, 3, 8, 7, 9 };
    int n = sizeof(a) / sizeof(a[0]);
    CocktailSort(a, n);
    printf("Sorted array :\n");
    printArray(a, n);
    return 0;
}
```

```
Microsoft Visual Studio Debug Console
Sorted array : 1 2 3 3 5 7 8 9
```

SORTED!

SLEEP SORT

```cpp
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>
#include <algorithm>
using namespace std;

template< typename It >
void sleepSort( const It& begin, const It& end )
{
    if ( begin == end )
        return;
    const auto [min, max] = minmax_element( begin, end );
    for ( auto it = begin; it != end; ++it )
    {
        thread( [min = *min, i = *it]
            {
                this_thread::sleep_for(chrono::milliseconds( i - min ) );
                cout << i << endl;
            } ).detach();
    }
    this_thread::sleep_for(chrono::milliseconds( *max - *min + 1 ) );
}

int main( int argc, char *argv[] )
{
    vector<int> v = { 20, 5, 15, 10, -5, };
    sleepSort(v.begin(), v.end() );

    return 0;
}
```

SORTED!

```
-5
5
10
15
20
```

OUR NEXT STEP!

- test out 8 different sized nearly, randomly and reverse sorted arrays.
- Measure their run times
- Compare and analyze
- Conclude findings!