Faculty of Engineering & Technology

Electrical & Computer Engineering Department

COMP2421-DATA STRUCTURES AND ALGORITHMS

Project#4

Sorting and Dynamic Programming

Prepared by: Ahmad Abu Saleem

Id:1201315

Instructor: Dr. Radi Jarrar

Section: 2

Date: 1/2/2023

1. Abstract:

The aim of this project :

Write three new algorithms for sorting and how to work.

Mention the algorithm properties: time complexity space complexity and various scenarios if the data is generated at random, if the data is sorted ascending, or if the data is sorted descending).

Also, answer these questions

a. What is Dynamic Programming?
b. Mention three problems that can be solved using Dynamic Programming.
c. Show an example of a code that solves a problem using Dynamic Programming and without using Dynamic Programming.

Table of contents:

1. Abstract ii

1.1 Table of contents: iii

2Sorting: 4

2.1 Count sort 4

2.2 Cyclic Sort 7

2.3 Bogo sort 10

3 Dynamic Programming 13

3.1 what is dynamic Programming 13

3.2 Three problems that were solved using Dynamic Programming……13

3.3 example of a code using Dynamic Programming and without……14

4 References……15

2 Sorting:

2.1 Count sort

count sort is an algorithm for sorting a group of items and it works well when the number of entered values and the input range are small. It is not an algorithm that relies on comparison. The idea of it is to count the number of occurrences of each item in the list and use this count to put each item in its correct place.

how it works

1) check the range of elements

2): Count the elements, and establish an array, with a size equal to the maximum element in the input array plus 1

3): sum and create a summation array

4):Check the last number in the input array. Move to the sum array index that matches this number in the input array. Decrease the value at this index in the sum array by one, and then add the value of the input array element.

Example

the input array {1, 3, 2, 8, 5, 1, 5, 1, 2, 7}

Step 1: Check Range
The range of elements is between 0 -9 .

Step 2: Count Elements
Create a count array with the size of the maximum element in the input array + 1

Count array = [0, 3, 2, 1, 0, 2, 0, 1, 1, 0]
Step 3: Summation array
Create a summation array by calculating the cumulative sum of the count array.
Summation array = [0, 3, 5, 6, 6, 8, 8, 9, 10, 10]

Step 4: Build the Sorted Array
Output array = [1, 1, 1, 2, 2, 3, 5, 5, 7, 8]

The Function code

```c
#include <stdio.h>
void countingSort(int array[], int size) {
  int output[10];
  // Find the largest element of the array
  int max = array[0];
  for (int i = 1; i < size; i++) {
    if (array[i] > max)
      max = array[i];
  }
  int count[10];
  // Initialize count array with all zeros.
  for (int i = 0; i <= max; ++i) {
    count[i] = 0;
  }
  // Store the count of each element
  for (int i = 0; i < size; i++) {
    count[array[i]]++;
  }
  // Store the cummulative count of each array
  for (int i = 1; i <= max; i++) {
    count[i] += count[i - 1];
  }
  for (int i = size - 1; i >= 0; i--) {
    output[count[array[i]] - 1] = array[i];
    count[array[i]]--;
  }
  for (int i = 0; i < size; i++) {
    array[i] = output[i];
  }
}
```

State the time and space complexity of various scenarios

Time complexity
Best case O(n+k)
Worst case O(n+k)
Avarge case O(n+k)
n the number of elements,
k the range of the input values.
Space complexity
O(k), where k is the range of the input values. This is because we need to create acont array of size k+1 to store the counts.
***Various* scenarios**
Sorted ascending
Time Complexity: O(n + k)
Space Complexity: O(k)
Sorted descending
Time Complexity: O(n + k)
Space Complexity: O(k)
Sorted Random
Time Complexity: O(n + k)
Space Complexity: O(k)

2.2 Cyclic sort

Cyclic sort is a quick algorithm whose basic idea is to organize elements by dividing the input array into circles by comparing and rotating each cycle to produce an ordered array, which is good for small data.

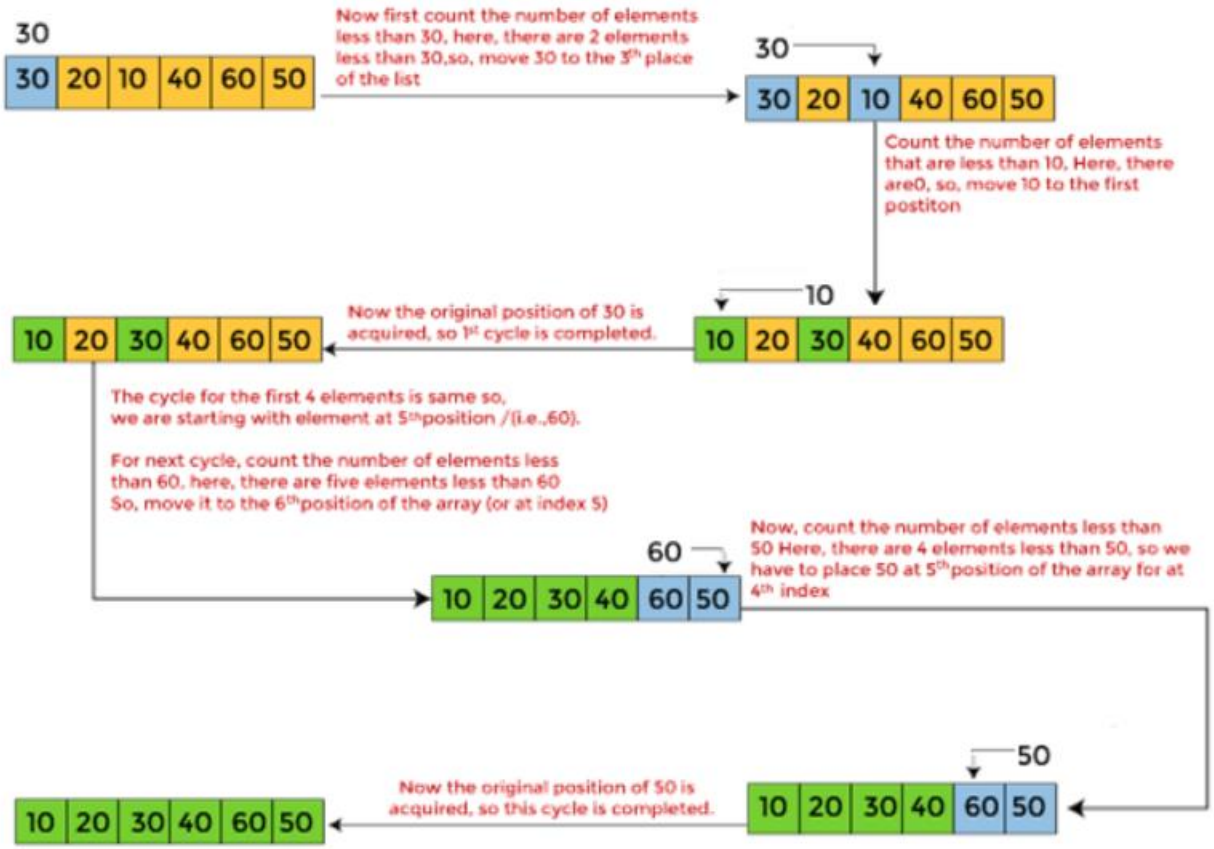
how it works

1 We need to find the correct place for each element by counting the number of smaller elements after it. If the sum is zero, leave it in its place, but if it is not zero, swap it with the element in this place.

2 After completing the switching process, repeat the first step with the item you switched to, then continue doing this process until everything is in its correct place.

Example

Let the elements of array are - **{30, 20, 10, 40, 60, 50}**



Note : The screenshot from https://www.javatpoint.com/cycle-sort and I put it in Reference

The Function code

```
void cycleSort(int a[], int n)
{
    int start, element, pos, temp, i;
    for (start = 0; start <= n - 2; start++) {
        element = a[start];
        pos = start;

        for (i = start + 1; i < n; i++)
            if (a[i] < element)
                pos++;
        if (pos == start)
            continue;
        while (element == a[pos])
            pos += 1;
        if (pos != start)
        {
            temp = element;
            element = a[pos];
            a[pos] = temp;
        }
        while (pos != start)
        {
            pos = start;
            for (i = start + 1; i < n; i++)
                if (a[i] < element)
                    pos += 1;
            while (element == a[pos])
                pos += 1;
            if (element != a[pos])
            {
                temp = element;
                element = a[pos];
                a[pos] = temp;
            }
        }
    }
}
```

state the time and space complexity of various scenarios

Time Complexity:
Best Case: O(n^2)
Worst Case: O(n^2)
Average Case: O(n^2)
n is the number of elements.

Space Complexity:
Space Complexity: O (1) (in place)

various scenarios
1.Sorted Ascending:
Time Complexity: O(n^2)
Space Complexity: O (1)

2. Sorted Descending:
Time Complexity: O(n^2)
Space Complexity: O (1)

3. Sorted Random:
Time Complexity: O(n^2)
Space Complexity: O (1)

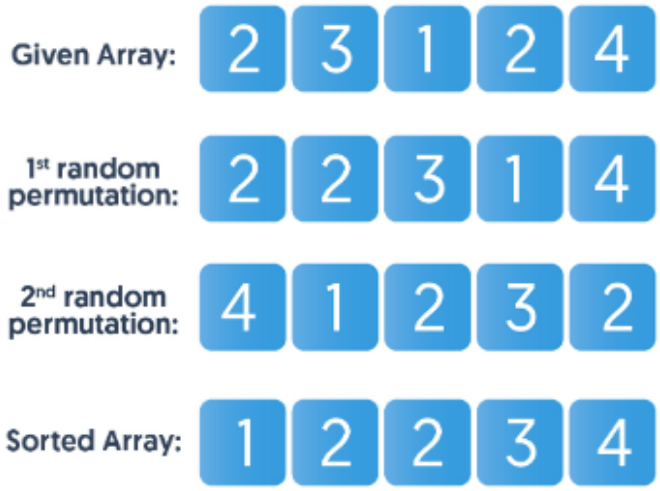
2.3 Bogo sort

Bogo Sort is an algorithm that depends on luck. It is called stupid sort. It is an algorithm that randomly shuffles the array and then checks whether the array is arranged or not, and if it is not arranged, the process will continue until it is arranged.

how it works

Arrange the elements randomly, then verify the order. If they are sorted correctly, it returns the array. Otherwise, it will rearrange the random order until it is arranged.

Example



The given array is {2,3,1,2,4} with the first attempt becoming {2,2,3,1,4} but it's not sorted so we do it again and became {4,1,2,3,2} but it's not sorted also so we do it again and became {1,2,2,3,4} and now its sorted.

The Function code

```c
// Function to check if an array is sorted
bool is_sorted(int *a, int n) {
    while (--n >= 1) {
        if (a[n] < a[n - 1])
            return false;
    }
    return true;
}
// Function to shuffle the elements of an array
void shuffle(int *a, int n) {
    int i, t, r;
    for (i = 0; i < n; i++) {
        t = a[i];
        r = rand() % n;
        a[i] = a[r];
        a[r] = t;
    }
}

void bogosort(int *a, int n) {
    while (!is_sorted(a, n))
        shuffle(a, n);
}
```

State the time and space complexity of various scenarios

Time Complexity:

Best Case: O(n)
Worst Case: O(n*n!)
Average Case: O(n*n!)

Space Complexity:

Space Complexity: O(1)

various scenarios

1. Sorted Ascending:

Time Complexity: O(n)
Space Complexity: O(1)

2. Sorted Descending:

Time Complexity: O(n*n!)
Space Complexity: O(1)

3. Sorted Random:

Time Complexity: O(n*n!)
Space Complexity: O(1)

3 Dynamic Programming:

a. What is Dynamic Programming?

It is one of the computer programming techniques, which is a big problem. To solve it, we divide the problem into several subsections, and we solve the subsections, save their results, and use them to solve the other sections. For example, we solve the first section and use its result to solve the second section, and use the second to solve the third, and so on. The closest example of it is It is Fibonacci

b. Mention three problems that can be solved using Dynamic Programming.

1) Fibonacci Sequence: The solution to the Fibonacci sequence using a simple recurrence its time complexity is O(n^2), but solving it using dynamic programming by dividing the results into smaller sections is better and easier, and the time complexity is less.

2) 0 - 1 Knapsack Problem: Solving the Problem using a Recurrence leads to a time complexity of O(2^n). This makes it less efficient for big problems but solving it using dynamic programming by dividing the results into smaller sections is better and easier, and the time complexity is less

3) Coin change problem: we can solve the coin change problem using dynamic programming which is An excellent and highly efficient method

C. Show an example of a code that solves a problem using Dynamic Programming and without using Dynamic Programming.

1. Fibonacci Sequence with recurrence

```
int fibonnacci( int n ){
      int answer;

      if(n == 0 || n == 1)
            answer = n;
      else
            answer = fibonnacci(n - 1) + fibonnacci(n - 2);

      return answer;
}
```

2. Fibonacci sequence with Dynamic Programming

```
#include <stdio.h>
int fib(int n) {
    int answer;
    if (n == 0 || n == 1) {
        answer = n;
    } else {
        int fibonacci[n+1];
        fibonacci[0] = 0;
        fibonacci[1] = 1;

        for (int i = 2; i <= n; i++) {
            fibonacci[i] = fibonacci[i-1] + fibonacci[i-2];
        }

        answer = fibonacci[n];
    }
    return answer;
}
```

4 References

https://www.programiz.com/dsa/counting-sort

https://www.studytonight.com/data-structures/counting-sort

https://www.youtube.com/watch?v=fl1PhC1z-Xc

https://www.javatpoint.com/counting-sort

https://www.javatpoint.com/cycle-sort

https://www.geeksforgeeks.org/cycle-sort/

https://www.w3resource.com/c-programming-exercises/searching-and-sorting/c-search-and-sorting-exercise-14.php

https://www.devmaking.com/learn/algorithms/bogo-sort/

https://www.interviewkickstart.com/learn/bogo-sort-algorithm

https://www.geeksforgeeks.org/bogosort-permutation-sort/

https://www.geeksforgeeks.org/dynamic-programming/

https://www.geeksforgeeks.org/coin-change-dp-7/

https://www.geeksforgeeks.org/introduction-to-dynamic-programming-data-structures-and-algorithm-tutorials/

https://www.javatpoint.com/dynamic-programming

COMP2421 Slide