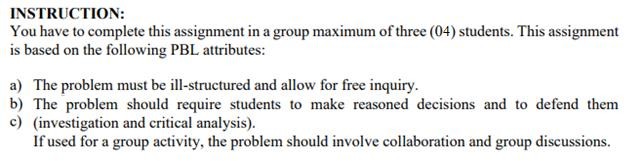
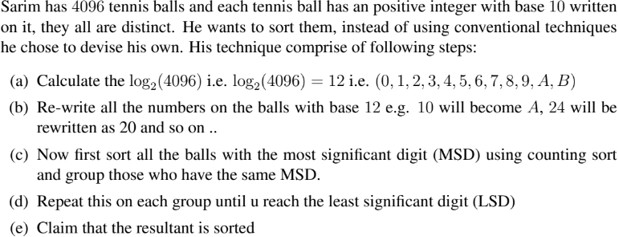
|  |  |
| --- | --- |
|  | **BAHRIA UNIVERSITY,**  **(Karachi Campus)**  *Department of Software Engineering*  **Assignment #03– Spring 2023**  **Problem Based Learning** |

|  |  |  |  |
| --- | --- | --- | --- |
| COURSE TITLE: | **D&AA** | COURSE CODE: | **CSC-321** |
| Class: | **BSE 4B** | Shift: | **Morning** |
| Course Instructor: | **ENGR. BUSHRA FAZAL KHAN** | Assignment Date: | **05-Jun-2023** |
| Max. Marks: | **4 Points(CLO3)** | Assignment Due: | **09- Jun -2023** |



PROBLEM





Name: Muhammad Shoaib Akhter Qadri Enrollment No: 02-131212-009

Name: Muhammad Shoaib Akhter Qadri Enrollment No: 02-131212-009

Name: Muhammad Shoaib Akhter Qadri Enrollment No: 02-131212-009

Answer No 1:

1. Given Value = 4096

Log2 (4096) = 12

Log3 (4096) = 7.57

Log4 (4096) = 6

Log5 (4096) = 5.16

Log6 (4096) = 4.67

Log7 (4096) = 4.27

Log8 (4096) = 4

i.e= {0,1,2,3,4,5,6,7,8,9,A,B}

1. Let ,

Array a[4096]

a=10 , b=11 , increment = 0;

for(i=1;i<a.length;i++)

{

if(i==a){

a[i]= ‘A’;

a+=10;

}

if(i==b){

a[i]= ‘B’;

b+=10;

}

else{

a[i]=increment;

increment ++;

}

}

1. There are different sorting Algorithms for this purpose in which Bucket and Counting Sort is best for this scenario and these sort all the balls.

* CountingSort(array,size)
* Find largest Element in array 🡪 max
* Initialize the Count with Zeros
* for(i🡨0 to size)
* find total count of each unique element and store the count to jth index.
* for i🡨1 to max
* find commulative sum and store it in count array itself.
* For j🡨size down to 1
* Restore the elements to array
* Decrease count of each element by 1.

1. While(LSD reach)

Repeat Step ‘C’

1. Console.WriteLine(“Sorted”);

Answer 2:

While Sanm's technique may work in theory, there is a more efficient and straightforward approach to sort the tennis balls based on their positive integer values. Here's a better design approach:

* Use a standard comparison-based sorting algorithm:

Rather than creating a custom technique involving logarithms and base conversions, it is more practical to use a well-known comparison-based sorting algorithm such as Quicksort, Mergesort, or Heapsort. These algorithms have been extensively studied and optimized, making them highly efficient for sorting large sets of data.

* Implement a stable sorting algorithm:

Stability in sorting means that the relative order of elements with equal values remains unchanged after the sorting process. It is important to maintain stability to avoid losing any meaningful order within the tennis balls. Algorithms like Mergesort and Timsort are inherently stable, while others can be modified to ensure stability.

* Utilize appropriate data structures:

Instead of counting sort, which requires a predefined range of values, consider using a suitable data structure such as a balanced binary search tree (e.g., AVL tree or Red-Black tree) or a priority queue (e.g., heap). These data structures provide efficient insertion and retrieval operations, which are essential for sorting the tennis balls.

* Compare and sort the positive integers directly:

Rather than converting the positive integers to a different base, perform the comparisons and sorting based on the original positive integer values themselves. This approach avoids the overhead of base conversion and allows for a more direct and efficient sorting process.

* Efficiency:

Above technique involves converting the base 10 numbers to base 12, which requires additional computations and string manipulations. This can be computationally expensive, especially when dealing with a large number of tennis balls. In contrast, algorithms like Quicksort and Merge Sort have a well-established efficiency of O(n log n) in the average case, which makes them more efficient for sorting tasks.

* Standardization:

above technique is specific to base 12 conversion and counting sort. While it may work for this particular scenario, it may not be as widely applicable to other sorting problems.

* Maintainability:

Quicksort and Merge Sort are well-known sorting algorithms that have been extensively studied and optimized over the years. They have clear and concise implementations, making the code more readable and maintainable.

* Reusability:

Quicksort or Merge Sort algorithms are widely supported and can be easily integrated into different applications.

By following these design approaches, we can achieve a more efficient and effective sorting algorithm for the given problem, ensuring that the tennis balls are sorted correctly.