**Advanced Algorithms**

**Exercise for Lecture 3**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** |  | **Student ID** |  |
| **Problem 1** |  | | |
| **Problem 2** |  | | |
| **Problem 3** |  | | |
| **Problem 4** |  | | |
| **Total Score** |  | | |
| **Notes** | Deadline: **2023-09-20 24:00**  Submission Format: ‘**Lecture3\_Name\_Student ID.docx**’, and please send to: **[chenlq1997@126.com](mailto:algorithms_23fall@163.com)**.  This assignment is meant to be an evaluation of your **individual** understanding coming into the course and should be completed **without collaboration** or outside help. | | |

**Problem 1.[25 points]** Please prove that COUNTING-SORT algorithm is stable.

**Solution:**

In counting sort, we place each element of the input array A in the output array B from back to front, the operation is:

for j ← n down to 1

Do B[C[A[ j]]] ← A[ j]

C[A[ j]] ← C[A[ j]] – 1

Assume that A[p] = A[q] and p<q, define:

r=C[A[j]]  while j=q  
 r′=C[A[j]]  while j=p

then r and r’ means the position of A[q] and A[p] in array B.

Because we traverse j from back to front, and every time we place a number in output array, we do

C[A[ j]] = C[A[ j]] – 1

so r′<r must be true, which means A[p] still precedes A[q] in output array B.

Therefore, counting-sort is stable.

**Problem 2.[25 points]** Please Explain why the worst-case running time for bucket sort is. What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time ?

**Solution:**

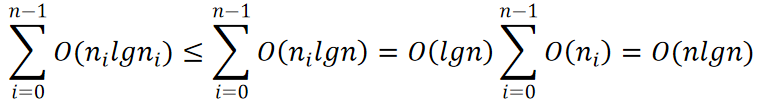
The worst-case running time for the bucket-sort is O(n2).

If the distribution of elements to sort is extremely uneven, then you might have a bucket k that has almost n elements in it. It will take O(n2) time to sort this bucket with insertion sort.

So in this case, the running time for the bucket-sort is:

In order to improve the worst-case running time of bucket sort to O(nlgn), we can simply replace the insertion sort used to sort each bucket with some worst case O(nlgn) sorting algorithm.

The sorting then takes time:

****

Now the total time of bucket sort is:

T(n)=Θ(n)+O(nlgn)=O(nlgn)

**Problem 3. [25 points]** Please sort the following numbers with radix sort algorithm. And you need to show every step. Only the result won’t get a full score.

784512 895623 486213 794613 579135 741963 369258 864795

**Solution:**

**784512 895623 486213 794613 579135 741963 369258 864795**

**784512 895623 486213 794613 741963 579135 864795 369258**

**784512 486213 794613 895623 579135 369258 741963 864795**

**579135 486213 369258 784512 794613 895623 864795 741963**

**741963 784512 794613 864795 895623 486213 579135 369258**

**741963 864795 369258 579135 784512 486213 794613 895623**

**369258 486213 579135 741963 784512 794613 864795 895623**

**Problem 4. [25 points]** Please sort the following numbers with shell sort algorithm. And you need to show every step. Only the result won’t get a full score. Please show how to group the data of each step.

8 9 11 7 25 3 15 6 4 0 1 5 2 13 12 18

**Solution:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **8** | **9** | **11** | **7** | **25** | **3** | **15** | **6** | **4** | **0** | **1** | **5** | **2** | **13** | **12** | **18** |
| **4** | **0** | **1** | **5** | **2** | **3** | **12** | **6** | **8** | **9** | **11** | **7** | **25** | **13** | **15** | **18** |
| **2** | **0** | **1** | **5** | **4** | **3** | **11** | **6** | **8** | **9** | **12** | **7** | **25** | **13** | **15** | **18** |
| **1** | **0** | **2** | **3** | **4** | **5** | **8** | **6** | **11** | **7** | **12** | **9** | **15** | **13** | **25** | **18** |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **11** | **12** | **13** | **15** | **18** | **25** |