# Homework 2

## Tianyi Zhang(tz168) 02/20/2018

1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 50 | 500 | 5000 | 10000 | 20000 |
| insertion | 719 | 60229 | 6314425 | 24951369 | 99496596 |
| shell | 344 | 11937 | 980451 | 3782365 | 14530626 |

With the help of h-sorting of 7,3,1. The array allows swapping far apart index and it will lead less comparison and easy for next step sort. After 7 and 3 sorting, the array has become “ordered” and easy for the final step 1 sorting which is insertion sort to comparison. Obvious, compared with unordered array, it needs less comparison for a part “ordered” array.

2. For this problem, because matrix A for convience is an ordered set of number, so the kendall Tau distance can be calculated only focus on matrix B. From the defination, by sorting matrix B, we can get the kendall Tau distance dirrectly. So in this problem, I choose to use merge sort and the time complexcity should be O(NlogN). Here is the data and figure.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| data size | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 |
| time(us) | 6052 | 16320 | 21909 | 34074 | 47070 | 61242 |

3. For this problem, I used counting sort method and the time complexity should be O(N). Because we already know the data kind and data size. It’s easy for us to make the sort process quickly. For counting method, it’s designed for the situation of some data repeat occurred and we also know the data kind and data repeat numbers. In this situation, one traverse could sort the data well. In my program, I input manually for data 1, 11, 111 and 1111. If you want to use this program for other problem, it needs to be modified.

4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| data1. | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 |
| Up-Bottom | 8954 | 19934 | 43944 | 96074 | 208695 | 450132 |
| Bottom-Up | 8954 | 19934 | 43944 | 96074 | 208695 | 450132 |

From the figure, we can see the comparison number of Up-bottom and Bottom-up quicksort are totally same. Because the only difference of these two methods is the start point. The core part of these two algorithms “Merge” are the same. When I write these two algorithms, I call the same “Merge” function. So the result of comparison must be the same. Generally, for the integrity of the performance of these two algorithms, bottom-up does not use recursion, for big data, it’s more efficient than up-bottom algorithm. Because recursion will consume lots of memory. For this problem, these two algorithm are almost the same performance.

5.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| data1 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 |
| merge | 8954 | 19934 | 43944 | 96074 | 208695 | 450132 |
| medianOf3 | 8104 | 18518 | 39814 | 87518 | 194789 | 415628 |
| cutoff-7 | 9981 | 20049 | 46259 | 96717 | 224915 | 415243 |
| mOf3&cut7 | 8227 | 18668 | 40223 | 88207 | 196024 | 418432 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| data0 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 |
| merge | 5120 | 11264 | 24576 | 53248 | 114688 | 245760 |
| medianOf3 | 9217 | 20481 | 45057 | 98305 | 212993 | 458753 |
| cutoff-7 | 524772 | 2098148 | 8390628 | 33558500 | \* | \* |
| mOf3&cut7 | 8065 | 18177 | 40449 | 89089 | 194561 | 421889 |

When data is ordered or partially order, for quicksort, it costs much more without using medianOf3 method, like Cutoff7, which means in order to prevent the worst case (already ordered) using medianOf3 is a good solution. In this situation, merge sort performs a little bit better than quicksort, but the total trends are same. In conclusion, quicksort with medianOf3 and cutoff is a good choice for all situation, but in small data size situation, merge sort performs better. If we need to sort a big size of data, quicksort performs better than merge sort.

6.

1.MergeSort(bottom-up) 2. Quicksort (standard, no shuffle) 3. Knuth shuffle 4.MergeSort(up-bottom)

5.Insertion 6.Heapsort 7.Selection 8.Quicksort(3-way,no shuffle)