Homework 5

Saturday, November 12, 2022 5:51 PM

Problem 1

a, The worst case running time of this sorting algorithm is the hash map is in reverse order, which the insertion sort took $O(n^2)$ running time. This happen when we get all elements in a bucket & apply insertion sort to that bucket.

The best case is when the hashmap is already sorted after all element added. This way don't have to execute inner loop but only run from first to last to check. Running time $O(n)$ when all elements in the bucket is sorted

b, Bucket Sort is better because we assume the data is uniformly distributed over multiple small bucket that might sorted. In those small number of elements in each bucket, insertion sort is faster ($O(n)$) than insertion sort in large range of array ($O(n \log n)$). Therefore, bucket sort is a better idea.

c, Using bucket sort is better than quicksort because bucket sort distributed elements into small buckets and processed independently. This will reduce the time running to time to sort each bucket times number of bucket.

Problem 2

The worst case for quicksort is $O(n^2)$ when the list is reverse order and each recursive call processes a list of size one less than previous list When the algorithm reaches the depth limit, switching to merge sort or heap sort that have worst case $O(n \log n)$ will reduce the time complexity. This significantly reduce time running. If d depth is reached $\Rightarrow O(n^2/d + n \log n) < O(n^2)$

$\Rightarrow$ The reasonable depth is $n/2$ because this depth helps quicksort not reaching worst case, reducing time complexity

Problem 3

The recursions stops when $\frac{n}{2^i} = k \Rightarrow 2^i = \frac{n}{k} \Rightarrow i = \log_2\left(\frac{n}{k}\right) = \log\left(\frac{n}{k}\right)$

$\Rightarrow$ Running time of recursion is $O(ni) = O(n \log(n/k))$ for n/k number of subarray.

For each subarray, using insertion sort have running time $O(k^2)$ (worst case) $\Rightarrow O(n/k \times k^2) = O(nk)$

$\Rightarrow$ Total running time $O(n \log(n/k)) + O(nk) = O(nk + n \log(n/k)) = O(n(k + \log(n/k)))$.

This method is better than insertion sort is because insertion sort has worst case running time $O(n^2)$ while this method is $O(n(k + \log(n/k)))$

Since $k + \log(n/k) < n \Rightarrow O(n(k + \log(n/k))) < O(n^2)$

Problem 4.

- Create left = 0, run from index 0
  right = n-1, run from index n-1
  While left < right, if (array[left] == true) swap left to right.
  decrement right --. else if (array[left] == tree) increment left.

$\Rightarrow$ Put false to left, true to right

Problem 5

a, For sorted array, the time complexity is $O(n \log n)$ because when array is sorted, the program still runs with average case. Elements around pivot arrange around pivot, so it's average case. It did not reach worst case because the worst case for quicksort is when pivot is leftmost of sorted array.

b, For reverse sorted array, the running time is $O(n \log n)$ because the program still runs average case when pivot in the middle, all other elements arrange around pivot. It did not reach worst case because the pivot is not rightmost of reverse order, where all elements on the left of pivot is larger than pivot.