WA4 - Assignment Group 16

Sravani Kirla - skirla
Matthew Heil - heilx069
Terrence Hanrahan - hanra028

1.

```
                                                        Running Time
  def processArray(A,k):
    counting = [0] * (k+1)                                  Θ(1)
    for j in range(0, len(A)):                              Θ(n)
        key = A[j]                                          Θ(n)
        counting[key] = counting[key] + 1                   Θ(n)
    for i in range(1, k+1):                                 Θ(k)
        counting[i] = counting[i] + counting[i - 1]         Θ(n)
    return counting                                         Θ(1)
                                                            Θ(n+k) → Total runtime to process
def query(counting,c,d):
    if c==0:                                                Θ(1)
        cCount = 0                                          Θ(1)
        dCount = counting[d]                                Θ(1)
    else:
        cCount = counting[c - 1]                            Θ(1)
        dCount = counting[d]                                Θ(1)
    intCount = dCount-cCount                                Θ(1)
    return intCount
```

Using part of counting-sort algorithm for preprocessing an Array ‘A’ gives count of each element of the input array ‘A’ as array ‘counting’ and by subtracting the count of ‘c-1’ from count of ‘d’ gives the number of integers in the range of c . . d. In preprocessing, running time is dependent on ‘k’ value and hence the run time will be Θ(n + k) and become Θ(n) when k is small.

2. BucketSort the array A into buckets where B[0] corresponds to strings of character length 0 in array A, B[1] corresponds to strings of character length 1 in array A, B[2] corresponds to strings of character length 2 in array A, etc. After all of the buckets are sorted, they are concatenated to an output array as follows:

> Take B[1], B[2] ... B[n] where n is the length of the longest string. B[0] elements always go first if there are elements in it. After that, the first element B[1] is added to output array. Then, if the first letters in any B[2], B[3], … B[n] elements are the same as the element just added to the output array, they are added to the output array. B[2] elements are added before B[3] elements, B[3] elements are added before B[4] elements and so on. This process is repeated until there are no more bucket elements left to be added to the output array. An example of this concatenation is below

B[1] = [‘a’, ‘b’, ‘c’], B[2] = [‘ab’, ‘bc’, ‘cd’], B[3] = [‘abc’, ‘bcd’, ‘cde’] Output = [‘a’,’ab’,’abc’] after looking at the first element in B[1] (‘a’) and searching for elements in B[2] and B[3] starting with the same letter. The next pass through yields Output = [‘a’, ‘ab’, ‘abc’, ‘b’, ‘bc’, ‘bcd’] and the final pass through gives the final outcome of Output = [‘a’, ‘ab’, ‘abc’, ‘b’, ‘bc’, ‘bcd’, ‘c’, ‘cd’, ‘cde’]. Finally, this output array is returned as the alphabetically sorted array.

The worst case run-time depends on insertion sort in the bucket sort, and the concatenation. Depending on the implementation, the concatenation step could be a double nested for loop that yields $\Theta(n^2)$ regardless since it needs to check for the same letter in each bucket after the one it is currently on. Also, insertion sorts average run-time is $\Theta(n^2)$ and even if the array is already sorted so insertion sort receives its best case, the algorithm still requires double nested for loops to check each element in a bucket and each bucket list. The worst-case run-time of this algorithm is $\Theta(2n^2)$ from the worst-case of the double nested for loops in forming the final output array and worst-case of insertion sort, or $\Theta(n^2)$.

3. Rod Cutting Optimization

length i	0	1	2	3	4	5	6
price p_i	0	2	3	7	8	10	15

Length = 1:
Optimal split: 1
Revenue = 2

Length = 2:
Optimal Split: 1-1
Revenue = 2+2 = 4

Length = 3:
Optimal Split: 3
Revenue = 7

Length = 4:
Optimal Split: 3-1
Revenue = 7+2 = 9

Length = 5:
Optimal Split: 3-1-1
Revenue = 7+2+2= 11

Length = 6:
Optimal Split: 6
Revenue = 15