Name: Suhail Basalama

Homework 4

8.2-4

We can achieve that with modifying the CountingSort algorithm slightly, which normally takes $\Theta(n+k)$. After preparing the new array C, which contains the accumulative occurrences of A's integers, we simply find the number of occurrences of integers between [a…b] in $O(1)$ with this formula:

$$\begin{cases} C[b]\ , & a = 0 \\ C[b] - C[a-1]\ , & a > 0 \end{cases}$$

The following is a python code to demonstrate the process:

```python
def countingSort(A,highest,a,b):

  C = [0]*(highest+1)
  B = [0]*len(A)

  for i in range(len(A)):
    C[A[i]]+=1

  for i in range(1,len(C)):
    C[i]+=C[i-1]

  if(a==0):
      return C[b]        #O(1) time complexity
  else:
      return C[b]-C[a-1] #O(1) time complexity
```

8.3-4

For a base-n integer x, the number of digits $d = \lceil \log_n(x+1) \rceil$. Applying this to the maximum possible number in the given array $(n^3 - 1)$

$\rightarrow d = \lceil \log_n(n^3 - 1 + 1) \rceil$

$\rightarrow d = 3$

Using RadixSort with n possible integers to sort for each digit we have:

$$\Theta\big(d(n+k)\big) = \Theta\big(3(n+n)\big) = \Theta(6n) = \Theta(n)$$

8.4-2

The worst-case running time would happen when we put all the n values in one bucket, and since we are using insertion sort to do this process for n values, it takes $O(n^2)$ on average. We can solve this by using any other $O(n \lg n)$ sorting algorithm like merge sort or quick sort to sort the values in the buckets.