**CSC 311 Fall 2019 Assignment 4**

**(2 points) Foundations of Algorithms, Chapter 2, Exercise Problem 2.**

So, 30 is the maximum number of comparisons.

**(4 points) Foundations of Algorithms, Chapter 2, Exercise Problem 5.**

The new strategy can only look left. That is, it can’t find a key that is at an index higher than l (lower bound). The number of comparisons is n, where n is the difference between l (lower bound) and the index of the key, if it is in the list.

**(4 points) Foundations of Algorithms, Chapter 2, Exercise Problem 7.**

//This algorithm finds the median of an array with i and n as inputs.

int findMedian(int[] a, int i,int n) {

Arrays.sort(a, i, i + n);

return a[i + n / 2]; // Return middle element

}

//This algorithm finds the smallest element at index with an array, left, right, and index as inputs

int smallestAtIndex(int[] a, int left, int right, int index) {

if (index > 0 && index <= right - left + 1) {

int n = right - left + 1 ; // Number of elements in a[left..right]

int i;

for (i = 0; i < n/5; i++) {

median[i] = findMedian(a, left + i \* 5, 5);

}

if (i\*5 < n) {

median[i] = findMedian(a, left + i \* 5, n % 5);

i++;

}

int medOfMed = median[i - 1];

if (i != 1)

medOfMed = smallestAtIndex(median, 0, i - 1, i / 2);

int pos = partition(a, left, right, medOfMed);

if (pos-left == index - 1)

return a[pos];

if (pos-left > index - 1) // If position is more, recur for left

return smallestAtIndex (a, left, pos - 1, index);

return smallestAtIndex (a, pos + 1, right, index - pos + left - 1);

}

return -1;

}

//This algorithm partitions with an array, left, right, and x as inputs

int partition(int[] a, int left, int right, int x) {

int i;

int temp;

for (i = left; i < right; i++) {

if (a[i] == x)

break;

}

temp = a[i];

a[i] = a[right];

a[right] = temp;

i = left;

for (int j = left; j < right; j++) {

if (a[j] <= x) {

temp = a[i];

a[i] = a[j];

a[j] = temp;

i++;

}

}

temp = a[i];

a[i] = a[right];

a[right] = temp;

return i;

}

main {

int[] testArray = {8, 1, 2, 45, 2, 11, 86, 21, 22, 11, 3, 4, 99, 34, 1294, 13, 1100};

int n = testArray.length;

int max = smallestAtIndex(testArray, 0, n-1, n);

}

}

**(4 points) Foundations of Algorithms, Chapter 2, Exercise Problem 11.**

function mergeSort(int[] a) {

if(a == null) {

return;

}

if(a.length > 1) {

int m = a.length / 2;

int[] left = new int[m];

for(int i = 0; i < m; i++) {

L[i] = a[i];

}

int[] right = new int[a.length - mid];

for(int i = m; i < a.length; i++) {

R[i - m] = a[i];

}

mergeSort(L);

mergeSort(R);

int i = 0;

int j = 0;

int k = 0;

while(i < L.length && j < R.length) {

if(L[i] < R[j]) {

a[k] = L[i];

i++;

}

else {

a[k] = R[j];

j++;

}

k++;

}

while(i < L.length) {

a[k] = L[i];

i++;

k++;

}

while(j < R.length) {

a[k] = R[j];

j++;

k++;

}

}

}

**(6 points) Foundations of Algorithms, Chapter 2, Exercise Problem 17.**

**(5 points) Foundations of Algorithms, Chapter 2, Exercise Problem 21.**

**(5 points) Foundations of Algorithms, Chapter 2, Exercise Problem 22.**

**(8 points) Foundations of Algorithms, Chapter 2, Exercise Problem 23.**

function partition(int a[], int low, int high) {

int pivot = a[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

if (a[j] <= pivot) {

i++;

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}

int t = a[i + 1];

a[i + 1] = a[high];

a[high] = t;

return i + 1;

}

function quickSort(int a[], int low, int high) {

if (low < high) {

int pivot = partition(a, low, high);

quickSort(a, low, pivot - 1);

quickSort(a, pivot + 1, high);

}

}

**(4 points) Foundations of Algorithms, Chapter 2, Exercise Problem 24.**

**(2 points) Foundations of Algorithms, Chapter 2, Exercise Problem 27.**

**(2 points) Foundations of Algorithms, Chapter 2, Exercise Problem 28.**

**(4 points) Foundations of Algorithms, Chapter 2, Exercise Problem 30.**

28948059

**(11 points) Foundations of Algorithms, Chapter 2, Exercise Problem 33.**

**(6 points) Foundations of Algorithms, Chapter 2, Exercise Problem 41.**

**(11 points) Foundations of Algorithms, Chapter 2, Exercise Problem 43.**

a)

Weigh two groups of 3 with the third group off to the side. If one of those is heavier, then you take any pair

from the heavier group and weigh them. Now, if one is heavier from weighing, you pick it. Otherwise you pick the third, not weighed coin. Similarly, if both groups are equal weight, do this process to the third, not weighed group.

b)

**(11 points) Foundations of Algorithms, Chapter 2, Exercise Problem 44.**

Function findRem(int x, int n, int p) {

If (n == 1) {

return x%p;

} else {

int root = n/2;

while (n > root) {

n--;

}

int p1 = findRem(x\*x, n, p);

int p2 = findRem(x\*x, n, p);

return (p1+p2)/2;

}

}

**(11 points) Foundations of Algorithms, Chapter 2, Exercise Problem 45.**