Ch.4

4.1 Write out the code for the earlier sum function.

#include <stdio.h>

#include <stdlib.h>

/\*\*4.1 Write out the code for the earlier sum function.\*\*/

int sum(int arr[],int len);

int main()

{

int ret=0;

int arr[]={3,4,5,5,6,7};

int len=sizeof(arr)/sizeof(arr[0]);

ret= sum(arr,len);

printf("sum=%i",ret);

return 0;

}

int sum(int arr[],int len)

{

int i=0;

if(len==0)

{

return 0;

}

else

{

return arr[len - 1] + sum(arr, len - 1);

}

}

4.2 Write a recursive function to count the number of items in a list.

int count(int arr[], int len)

{ if (len == 0) { return 0; }

else {

return 1 + count(arr, len - 1);

}

}

4.3 Find the maximum number in a list.

int Find\_max(int arr[], int len, int max) {

if (len == 0) {

return max;

}

if (arr[len - 1] > max){

max = arr[len - 1];

}

return Find\_max(arr, len - 1, max);

}

4.4 Remember binary search from chapter 1? It’s a divide-and-conquer algorithm, too. Can you come up with the base case and recursive case for binary search?

int binary\_search(int arr[],int first ,int last,int Snumber){

int midpoint= (first+last)/2 ;

// Recursive cases

if(Snumber==arr[midpoint]){

return midpoint;

}

else if(Snumber>arr[midpoint]){

return binary\_search(arr,midpoint+1 ,last,Snumber);

}

else if(Snumber>arr[midpoint]) {

return binary\_search(arr,first,midpoint-1,Snumber);

}

// Base case: search range is empty

if (first > last) {

return -1; // Element is not found

}

}

How long would each of these operations take in Big O notation?

4.5 Printing the value of each element in an array.

O(n)

4.6 Doubling the value of each element in an array.

O( n)

4.7 Doubling the value of just the first element in an array.

O(1)

4.8 Creating a multiplication table with all the elements in the array. So if your array is [2, 3, 7, 8, 10], you first multiply every element by 2, then multiply every element by 3, then by 7, and so on.

O(n^2)