**(1 point) Foundations of Algorithms Chapter 1, Exercise Problem 15.**

**(2 points) Foundations of Algorithms Chapter 1, Exercise Problem 22.**

**(3 points) Foundations of Algorithms Chapter 1, Exercise Problem 25.**

Old: instance size = 1000; time = 1 min

New: 1000 times faster.

Old: 1000 = n = 1 min

New: 1000\*1000 = n = 1 min

Instance size: 1000\*1000 = 1000000

Input size: 1000

Old: 1000 = ; /\*An input of size 10 causes an instance size of 1000. This takes the old machine 1 minute to do.\*/

New: 1000\*1000 = = /\*An input of size 100 causes an instance size of 100000. This takes the new machine 1 minute to do.\*/

Instance size: 1000\*1000 = 1000000

Input size: 100

Old: 1000 = ; /\*An input of size 3 causes an instance size of 1000. This takes the old machine 1 minute to do.\*/

New: 1000\*1000 = = /\*An input of size 6 causes an instance size of 100000. This takes the new machine 1 minute to do.\*/

Instance size: 1000\*1000 = 1000000

Input size: 6

**(5 points) Foundations of Algorithms Chapter 1, Exercise Problem 27.**

For simplicity in evaluating limits, I will be using ln(n) instead of lg(n). All log base b functions of n are in the same complexity.

Therefore, is O(n).

Therefore, n is O().

Therefore, is O().

Therefore, is ().

Therefore, is ().

**(2 points) Foundations of Algorithms Chapter 1, Exercise Problem 32.**

|  |  |  |
| --- | --- | --- |
| i | j | reps or T(n) |
| 1 | 1 | 1 |
| 2 | 2,1 | 2 |
| 3 | 4,2,1 | 3 |
| 4 | 4,2,1 | 3 |
| 5 | 8,4,2,1 | 4 |
| … | … | … |
|  |  |  |

**(4 points) Foundations of Algorithms Chapter 1, Exercise Problem 33.**

Problem: Sort a list into two sublists such that the sum of each sublist is maximized.

Input(s): An integer array with length n.

void maxDiff(int [] a) {

int n = a.length;

for (int i = 0; i < n-1; I++) {

for (int j = 0; j < n-i-1; j++) {

if (a[j] > a[j+1]) {

int t = a[j];

a[j] = a[j+1];

a[j+1] = t;

}

}

}

int [] upper = new int[n/2];

int [] lower = new int[n/2];

int cnt = 0;

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

if ( i <= n/2)

lower[i] = a[i];

else

upper[cnt++] = a[i];

}

//lower sublist

print(lower);

//upper sublist

print(upper);

//should have maximized difference in sum at this point.

}

**(2 points) Foundations of Algorithms Chapter 1, Exercise Problem 34.**

|  |  |  |
| --- | --- | --- |
| i | j | reps or T(n) |
| n | n | 1 |
|  |  | 2 |
| … | … | … |
|  |  |  |

**(3 points) Foundations of Algorithms Chapter 1, Exercise Problem 35.**

1. 32+25 = 57
2. T(n) = 2n //ignoring basic, linear operations: 2n+3
3. If you combine the operations into a single for loop, you get T(n) = n, but the big Oh is still n.

**(5 points) Foundations of Algorithms Chapter 1, Exercise Problem 36.**

1. Constant: index, first step of each for loop, and second step of last for loop to find the first element of first row is equal to the second element of the first row. This is 6 operations.
2. n^4
3. Set<Integer> hashset = new HashSet<>();

for (int i = 0; i < n-1; I++) {

for (int j = 0; j < n-i-1; j++) {

if(hashset.contains(a[i][j])) {

return 1;

}

}

}

return 0;

1. There are no duplicate elements
2. There is at least one duplicate element

**(4 points) Foundations of Algorithms Chapter 1, Exercise Problem 37.**

public static int find\_Remainder(int x, int n, int p) {

if (n == 1) {

return x % p;

} else {

return rem(x, n / 2, p);

}

}

**(4 points) Foundations of Algorithms Chapter 1, Exercise Problem 41.**

public static int findMinSum(int arr[], int i, int calcedSum , int sum) {

        if (i == 0) {

            return Math.abs((sum - calcedSum ) - calcedSum );

        }

        return Math.min(findMinSum (arr, i - 1, calcedSum  + a [i-1], sum),

                                 findMinSum (a, i-1, calcedSum , sum));

    }

public static int findMin(int[] a, int n) {

        int sum = 0;

        for (int i = 0; i < n; i++) {

            sum += a[i];

       }

        return findMinSum(a, n, 0, sum);

}

**(4 points) Foundations of Algorithms Chapter 1, Exercise Problem 42.**

**(4 points) Recursive Programming Chapter 2, Exercise 2.1.**

The size of the problem is the lg(n)

Size = lg(n)

**(3 points) Recursive Programming Chapter 2, Exercise 2.2.**

public static int sumNums(int n) {

if (n == 0)

return 0;

else if (n == 1)

return 1;

else if (n == 2)

return 3;

else {

if (n%2 == 1)

return (3\*(sumNums((n-1)/2))+sumNums((n+1)/2));

else

return (3\*(sumNums(n/2))+sumNums((n/2)-1));

}

}

**(2 points) Recursive Programming Chapter 2, Exercise 2.3.**

public static int sumNums(int n) {

if (n == 1)

return 1;

else {

if (n%2 == 1)

return (2\*(sumNums((n-1)/2))+((n+1)/2)\*((n+1)/2));

else

return (2\*(sumNums(n/2))+(n/2)\*(n/2));

}

}

**(2 points) Recursive Programming Chapter 2, Exercise 2.6.**

public static int getMax(int[] a, int n) {

if (n < 1)

return a[0];

else

return Math.max(getMax(a, n-1), a[n]);

}

Where n is a.length – 1.

**(3 points) Recursive Programming Chapter 3, Exercise 3.1.**

**(3 points) Recursive Programming Chapter 3, Exercise 3.2.**

For simplicity in evaluating limits, I will be using ln(n) instead of log(n). All log base b functions of n are in the same complexity.

**(3 points) Recursive Programming Chapter 3, Exercise 3.3.**

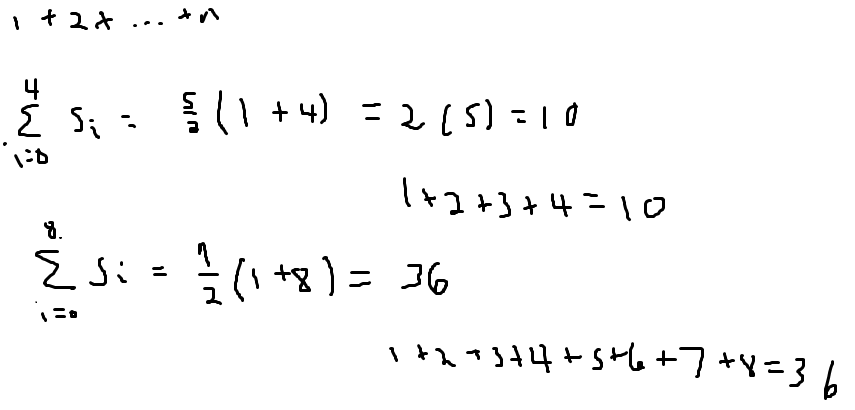
**(2 points) Recursive Programming Chapter 3, Exercise 3.5.**

This reads as the sum of the first n odd integers is n\*n.

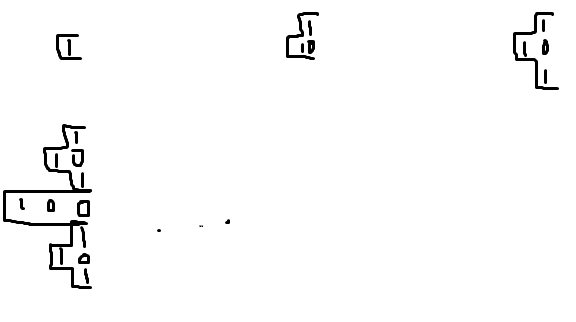
For ex.) Sum of first 3 odd integers (1+3+5 = 9; 3\*3 = 9).

**(3 points) Recursive Programming Chapter 3, Exercise 3.6.**

**(2 points) Recursive Programming Chapter 3, Exercise 3.7.**

****

**(2 points) Recursive Programming Chapter 3, Exercise 3.8.**

****

Number of bits =