Bit Questions

Basic Bit Level

## Big O

### Properties

What is the asymptotic running time of the following?

public static int SearchRecursive(int[] arr, int searchKey)

{

if (arr == null) throw new ArgumentNullException();

return SearchRecursive(arr, 0, arr.Length - 1, searchKey);

}

private static int SearchRecursive(int[] arr, int lo, int hi, int searchKey)

{

// The search key is not in the array. Return the complement of the index

// at which it should be inserted.

if (lo > hi) return ~lo;

int median = lo + (hi - lo) / 2;

int comparisonResult = arr[median].CompareTo(searchKey); ;

// a direct hit

if (comparisonResult == 0) return median;

return comparisonResult < 0

? SearchRecursive(arr, median + 1, hi, searchKey)

: SearchRecursive(arr, lo, median - 1, searchKey);

}

The running time is

Why do we not care about the base of the logarithm?

Because ==

So and differ by a constant factor and we don’t worry about constant factors in asymptotic notation

What is the asymptotic running time of the following?

public static int FibonacciRecursive(int n)

{

if (n == 0)

return 0;

if (n == 1)

return 1;

return FibonacciRecursive(n - 1) + FibonacciRecursive(n - 2);

}

The running time is where

What is the running time of the following function and what does it do?

public static int Function(int n)

{

if (n == 1)

return 1;

return Function(n - 1) + Function(n - 1);

}

The running time is or exponential. It calculates

What is the asymptotic running time of the following?

public static int Function2(int n)

{

int res = 0;

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

res += i;

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

res += i;

return res;

}

The running time is We ignore the fact it is 2n as we drop the constant factors

What is the asymptotic running time of the following?

public static int PairCount(int[] a)

{

int count = 0;

for (int i = 0; i < a.Length; i++)

{

for (int j = i + 1; j < a.Length; j++)

{

if (a[i] == a[j]) count++;

}

}

return count;

}

The running time is . Remember that the sum of the first n integers is given by

s=

We can write 2s as

So in our we are replacing n with n-1

In our asymptotic notation we call this by dropping the lower order terms.

What is the asymptotic running time of the following?

public static void PrintPairs(int[] a, int[] b)

{

for (int i = 0; i < a.Length; i++)

{

for (int j = 0; j < b.Length; j++)

{

Console.WriteLine($"({a[i]},{b[j]})");

}

}

}

where x is the number of elements in a and y is the number of elements in b

What is the asymptotic running time of the following?

public static void PrintPairsManyTimes(int[] a, int[] b)

{

for (int i = 0; i < a.Length; i++)

{

for (int j = 0; j < b.Length; j++)

{

for (int k = 0; k < 10; k++)

{

Console.WriteLine($"({a[i]},{b[j]})");

}

}

}

}

where x is the number of elements in a and y is the number of elements in b which is of course just

What is the asymptotic running time of the following?

public void ReverseArray(int[] a)

{

for (int i = 0; i < a.Length/2; i++)

{

int temp = a[i];

a[i] = a[a.Length-1-i];

a[a.Length-1-i] = temp;

}

}

We ignore the constant factor of

What is the asymptotic running time of sorting each string in an array and then sorting the array itself?

If we let the number of strings in the array be n and the length of each string be l then sorting each string takes We have to do this n time so we get The sorting of the array itself is but each string comparison requires l character compares in the work case so it is actually Adding the two thing together we obtain

What is the asymptotic running time of the following code?

public static bool IsPrimeNaive(int x)

{

if (x <= 1) return false;

for (int i = 2; i < x; i++)

{

if (x % i ==0)

return false;

}

return true;

}

The runtime is then

What is the asymptotic running time of the following code?

public bool IsPrimeUsingSquareRoot(int n)

{

if (n < 2)

return false;

if (n == 2)

return true;

// The definition of a prime is an integer x

// which is not exactly divisible by any

// number other than itself and one. If a

// number x is not prime it can be written as

// the product of two factors a x b. If both

// a and b were greater than the square root of

// x then a x b would also be greater than x and hence

// a x b is not x. SO testing all factors up to floor(root(x))

// is sufficient as if one factor is floor(root(x)) the other factor must

// be less than that

// hence test the n-2 integers from

// 2,..., Floor(Root(N))

return Enumerable.Range(2, (int)Math.Floor(Math.Sqrt(n)))

.All(i => n % i > 0);

}

The runtime is then

What is the asymptotic running time of the following code?

public static int Factorial(int x)

{

if (x ==0) return 1;

return x \* Factorial(x-1);

}

The running time is simple