Algorithm Questions

BIG O

SEARCHING

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 $O(\log n)$

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

Because $\log_a x = \frac{\log_b x}{\log_b a} = \log_b x \frac{1}{\log_b a} =$

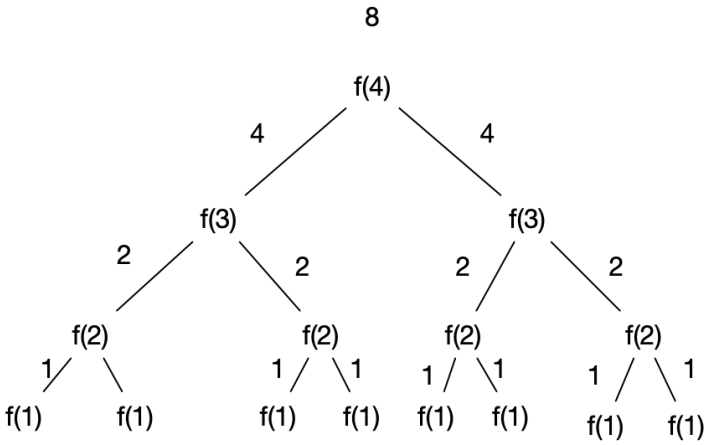
So $\log_a x$ *and* $\log_b x$ *differ by a constant factor and we don't worry about constant factors in asymptotic notation*

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);
}
```

Look at the call graph of the specific case of Function(4). We get the total run time by summing the number of calls at each level. 1+2+4+8. As we move down from one level to the other each level has double the number of calls as the level before. In our case we have $1 + 2 + 4 + 8 = 2^0 + 2^1 + 2^2 + 2^3 = 2^4 - 1$

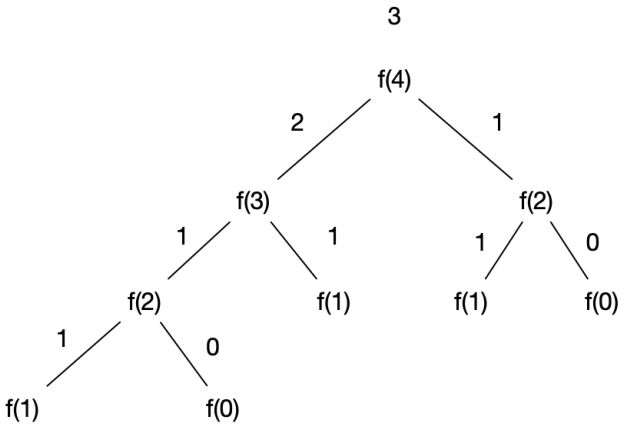


The running time is $O(2^{n-1}) = \frac{O(2^n)}{2} = O(2^n)$ *Why? Because* $2^{n-1} = \frac{1}{2^{n-1}2}$ *and we can ignore constant factors.*

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 call graph for fibonacci is very similar to the previous question so $O(2^n)$ *is a correct upper bound. It is possible to prove a tighter upper bound as* $O(\phi^n)$ *where* $\phi = (1 + \sqrt{5})/2$

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);
}
```

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 $O(n)$ 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 $O(n^2)$. Remember that the sum of the first n integers is given by

$s=1+2+\cdots(n-2)+(n-1)+n$

We can write 2s as

$$1+2+\cdots(n-2)+(n-1)+n$$

$$n+(n-1)+(n-2)+\cdots+2+1$$

$$2s=n(n+1)\therefore s=\frac{n(n+1)}{2}$$

So in our we are replacing n with n-1

$$s=\frac{(n-1)((n-1)+1)}{2}=\frac{(n-1)n}{2}$$

In our asymptotic notation we call this $O(n^2)$ 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]})");
		}
	}
}
```

$O(xy)$ 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]})");
			}
		}
	}
}
```

$O(10xy)$ where x is the number of elements in a and y is the number of elements in b which is of course just $O(xy)$

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;
	}
}
```

$O(n)$ We ignore the constant factor of $\frac{n}{2}$

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 $O(l \log l)$ *We have to do this n time so we get* $O(n \times l \log l)$ *The sorting of the array itself is* $O(n \log n)$ *but each string comparison requires l character compares in the work case so it is actually* $O(l \times n \times log\, n)$ *Adding the two thing together we obtain*

$$O(n \times l \log l + l \times n \times log\, n) = O(nl(\log l + log\, n) =$$

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 $O(x)$

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 $O(\sqrt{n})$

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 $O(x)$

Searching

Iterative Binary Search

Implement Iterative Binary Search and state its asymptotic run time?

```
public int SearchIterative<T>(IList<T> arr, T val)
  where T : IComparable<T>
{
    if (arr == null)
          throw new ArgumentNullException();

    int loIdx = 0;
    int hiIdx = arr.Count - 1;
    while (loIdx <= hiIdx)
    {
          int miIdx = loIdx + (hiIdx - loIdx) / 2;
          int comp = val.CompareTo(arr[miIdx]);

          if (comp == 0)
                return miIdx;

          if (comp > 0)
                loIdx = miIdx + 1;
          else
                hiIdx = miIdx - 1;
    }

    return ~loIdx;
}
```

The runtime is O(logN)