

1、分别简述队列、栈、堆的区别？

队列是先进先出：就像一条路，有一个入口和一个出口，先进去的就可以先出去。而栈就像一个箱子，后放的在上边，所以后进先出。堆是在程序运行时，而不是在程序编译时，申请某个大小的内存空间。即动态分配内存，对其访问和对一般内存的访问没有区别。

栈(Stack)是操作系统在建立某个进程时或者线程为这个线程建立的存储区域。在编程中，例如C/C++中，所有的局部变量都是从栈中分配内存空间，实际上也不是什么分配，只是从栈顶向上用就行，在退出函数的时候，只是修改栈指针就可以把栈中的内容销毁，所以速度最快。

堆(Heap)是应用程序在运行的时候请求操作系统分配给自己内存，一般是申请/给予的过程。由于从操作系统管理的内存分配所以在分配和销毁时都要占用时间，所以用堆的效率低的多!但是堆的好处是可以做的很大，C/C++对分配的Heap是不初始化的。

2、用JavaScript实现二分法查找

二分法查找，也称折半查找，是一种在有序数组中查找特定元素的搜索算法。查找过程可以分为以下步骤：

(1)首先，从有序数组的中间的元素开始搜索，如果该元素正好是目标元素(即要查找的元素)，则搜索过程结束，否则进行下一步。

(2)如果目标元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半区域查找，然后重复第一步的操作。(3)如果某一步数组为空，则表示找不到目标元素。

```
1 //二分搜索
2 //A为已按"升序排列"的数组，x为要查询的元素
3 //返回目标元素的下标
4 function binarySearch(A, x) {
5     var low = 0, high = A.length - 1;
6     while (low <= high) {
7         var mid = Math.floor((low + high) / 2); //下取整
8         if (x == A[mid]) {
9             return mid;
10        }
11        if (x < A[mid]) {
12            high = mid - 1;
13        }
14        else {
15            low = mid + 1;
16        }
17    }
18    return -1;
19 }
```

3、用JavaScript实现数组快速排序

关于快排算法的详细说明，可以参考阮一峰老师的文章快速排序 “快速排序”的思想很简单，整个排序过程只需要三步：

- (1)在数据集之中，选择一个元素作为“基准”(pivot)。
- (2)所有小于“基准”的元素，都移到“基准”的左边;所有大于“基准”的元素，都移到“基准”的右边。
- (3)对“基准”左边和右边的两个子集，不断重复第一步和第二步，直到所有子集只剩下一个元素为止。

方法一(尽可能不用js数组方法)：

```
1 function quickSort(arr){
2   qSort(arr,0,arr.length - 1);
3 }
4 function qSort(arr,low,high){
5   if(low < high){
6     var partKey = partition(arr,low,high);
7     qSort(arr,low, partKey - 1);
8     qSort(arr,partKey + 1,high);
9   }
10 }
11 function partition(arr,low,high){
12   var key = arr[low]; //使用第一个元素作为分类依据
13   while(low < high){
14     while(low < high && arr[high] >= arr[key])
15       high--;
16     arr[low] = arr[high];
17     while(low < high && arr[low] <= arr[key])
18       low++;
19     arr[high] = arr[low];
20   }
21   arr[low] = key;
22   return low;
23 }
```

方法二（使用js数组方法）：

```

1 function quickSort(arr){
2   if(arr.length <= 1) return arr;
3   var index = Math.floor(arr.length/2);
4   var key = arr.splice(index,1)[0];
5   var left = [],right = [];
6   arr.forEach(function(v){
7     v <= key ? left.push(v) : right.push(v);
8   });
9   return quickSort(left).concat([key],quickSort(right));
10 }

```

方法三：递归法

```

1 function quickSort(arr){
2   if(arr.length<=1){
3     return arr;//如果数组只有一个数，就直接返回；
4   }
5
6   var num = Math.floor(arr.length/2);//找到中间数的索引值，如果是浮点
  数，则向下取整
7
8   var numValue = arr.splice(num,1);//找到中间数的值
9   var left = [];
10  var right = [];
11
12  for(var i=0;i<arr.length;i++){
13    if(arr[i]<numValue){
14      left.push(arr[i]);//基准点的左边的数传到左边数组
15    }
16    else{
17      right.push(arr[i]);//基准点的右边的数传到右边数组
18    }
19  }
20
21  return quickSort(left).concat([numValue],quickSort(right));//
  递归不断重复比较
22  }
23
24  alert(quickSort([32,45,37,16,2,87]));//弹出“2,16,32,37,45,87”

```

4、编写一个方法 求一个字符串的字节长度

假设：一个英文字符占用一个字节，一个中文字符占用两个字节

```

1 function GetBytes(str){
2

```

```

3      var len = str.length;
4
5      var bytes = len;
6
7      for(var i=0; i<len; i++){
8
9          if (str.charCodeAt(i) > 255) bytes++;
10
11      }
12
13      return bytes;
14
15  }
16
17  alert(GetBytes("你好,as"));

```

5、找出下列正数组的最大差值

输入 [10,5,11,7,8,9]

输出 6

```

1  function getMaxProfit(arr) {
2
3      var minPrice = arr[0];
4      var maxProfit = 0;
5
6      for (var i = 0; i < arr.length; i++) {
7          var currentPrice = arr[i];
8
9          minPrice = Math.min(minPrice, currentPrice);
10
11          var potentialProfit = currentPrice - minPrice;
12
13          maxProfit = Math.max(maxProfit, potentialProfit);
14      }
15
16      return maxProfit;
17  }

```

6、判断一个单词是否是回文?

什么是回文?

回文是指把相同的词汇或句子，在下文中调换位置或颠倒过来，产生首尾回环的情趣，叫做回文，也叫回环。比如 mamam redivider。

```
1 function checkPalindrom(str) {
2     return str == str.split('').reverse().join('');
3 }
```

```
1 // while loop
2 const isPalindromicB = (w) => {
3
4     let len = w.length;
5     // 感谢 @拉比克魔王 的指点
6     let start = Math.ceil(len / 2);
7     while (start < len) {
8         if (w[start] !== w[len - start - 1]) {
9             return false;
10        }
11        start++;
12    }
13    return true;
14 };
```

7、如何消除一个数组里面重复的元素？

基本数组去重

```
1 Array.prototype.unique = function(){
2     var result = [];
3     this.forEach(function(v){
4         if(result.indexOf(v) < 0){
5             result.push(v);
6         }
7     });
8     return result;
9 }
```

利用hash表去重，这是一种空间换时间的方法

```

1 Array.prototype.unique = function(){
2   var result = [],hash = {};
3   this.forEach(function(v){
4     if(!hash[v]){
5       hash[v] = true;
6       result.push(v);
7     }
8   });
9   return result;
10 }

```

上面的方法存在一个bug，对于数组[1,2,'1','2',3]，去重结果为[1,2,3]，原因在于对象对属性索引时会进行强制类型转换，arr['1']和arr[1]得到的都是arr[1]的值，因此需做一些改变：

```

1 Array.prototype.unique = function(){
2   var result = [],hash = {};
3   this.forEach(function(v){
4     var type = typeof(v); //获取元素类型
5     hash[v] || (hash[v] = new Array());
6     if(hash[v].indexOf(type) < 0){
7       hash[v].push(type); //存储类型
8       result.push(v);
9     }
10  });
11  return result;
12 }

```

先排序后去重

```

1 Array.prototype.unique = function(){
2   var result = [this[0]];
3   this.sort();
4   this.forEach(function(v){
5     v !== result[result.length - 1] && result.push(v); //仅与result最后一个元素比较
6   });
7 }

```

8、统计字符串中字母个数或统计最多字母数

输入：afjghdfraaaasdenas

输出：a

```

1 function findMaxDuplicateChar(str) {
2   if(str.length == 1) {

```

```

3      return str;
4  }
5  let charObj = {};
6  for(let i=0;i<str.length;i++) {
7      if(!charObj[str.charAt(i)]) {
8          charObj[str.charAt(i)] = 1;
9      }else{
10         charObj[str.charAt(i)] += 1;
11     }
12 }
13 let maxChar = '',
14 maxValue = 1;
15 for(var k in charObj) {
16     if(charObj[k] >= maxValue) {
17         maxChar = k;
18         maxValue = charObj[k];
19     }
20 }
21 return maxChar;
22 }

```

9、随机生成指定长度的字符串

```

1  function randomString(n) {
2      let str = 'abcdefghijklmnopqrstuvwxyz9876543210';
3      let tmp = '',
4          i = 0,
5          l = str.length;
6      for (i = 0; i < n; i++) {
7          tmp += str.charAt(Math.floor(Math.random() * l));
8      }
9      return tmp;
10 }

```

10、写一个isPrime()函数，当其为质数时返回true，否则返回false。

首先，因为JavaScript不同于C或者Java，因此你不能信任传递来的数据类型。如果面试官没有明确地告诉你，你应该询问他是否需要做输入检查，还是不进行检查直接写函数。严格上说，应该对函数的输入进行检查。

第二点要记住：负数不是质数。同样的，1和0也不是，因此，首先测试这些数字。此外，2是质数中唯一的偶数。没有必要用一个循环来验证4,6,8。再则，如果一个数字不能被2整除，那么它不能被4，6，8等整除。因此，你的循环必须跳过这些数字。如果你测试输入偶数，你的算法将慢2倍（你测试双倍数字）。可以采取其他一些更明智的优化手段，我这里采用的是适用于大多数情况的。例如，如果一个数字不能被5整除，它也不会被5的倍数整除。所以，没有必要检测10,15,20等等。

最后一点，你不需要检查比输入数字的开方还要大的数字。我感觉人们会遗漏掉这一点，并且也不会因此而获得消极的反馈。但是，展示出这一方面的知识会给你额外加分。

现在你具备了这个问题的背景知识，下面是总结以上所有考虑的解决方案：

```
1 function isPrime(number) {
2   // If your browser doesn't support the method Number.isInteger of
  ECMAScript 6,
3   // you can implement your own pretty easily
4   if (typeof number !== 'number' || !Number.isInteger(number)) {
5     // Alternatively you can throw an error.
6     return false;
7   }
8   if (number < 2) {
9     return false;
10  }
11  if (number === 2) {
12    return true;
13  } else if (number % 2 === 0) {
14    return false;
15  }
16  var squareRoot = Math.sqrt(number);
17  for(var i = 3; i <= squareRoot; i += 2) {
18    if (number % i === 0) {
19      return false;
20    }
21  }
22  return true;
23 }
```

11、有100格台阶，可以跨1步可以跨2步，那么一个有多少种走法

本质是斐波那契算法

```
1 var Stairs = new step();
2 function step(){
3   this.n1=1;
4   this.n2=2;
5   this.total=100;
```



```

6     this.getFunction = getFunction;
7 }
8 function getFunction(){
9     for(i=2;i<this.total;i++){
10         res = this.n1 + this.n2;
11         this.n1 = this.n2;
12         this.n2 = res;
13     }
14     return res;
15 }
16 var totalStairs = Stairs.getFunction();
17 alert(totalStairs)

```

**12、请用算法实现，从给定的无序、不重复的数组data中，取出n个数，使其相加和为sum。并给出算法的时间/空间复杂度。
(找到一个解即可)**

```

1  /*
2  背包问题
3  有N件物品和一个容量为V的包，时间和空间复杂度均为O(N*V)
4  由于本题只需要找到一个解，所以空间复杂度为O(V)
5  */
6
7  function getResult(data,n,sum){
8      if(n==0 &&sum==0){
9          return true
10     }
11     if(n<0){
12         return false
13     }
14     if(n>0){
15         for(var i = 0; i < data.length; i++){
16
17             var temp = data.slice(i+1,data.length);
18
19             return getResult(temp,n-1,sum-data[i]) || getResult(temp,n,sum);
20         }
21     }
22 }
23
24 };
25 console.log(getResult([1,2,3,4,5,6,7],4,19))
26

```