### Quicksort

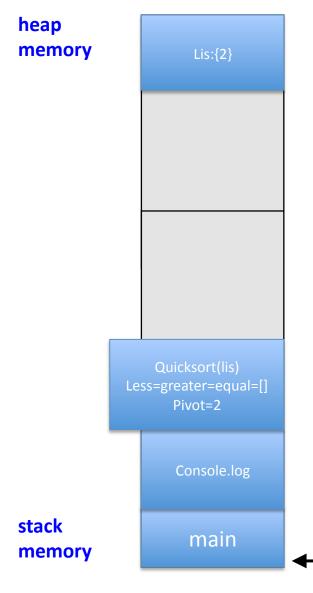
R1.2/R1.3

## heap memory Lis:{2} New array() stack main memory

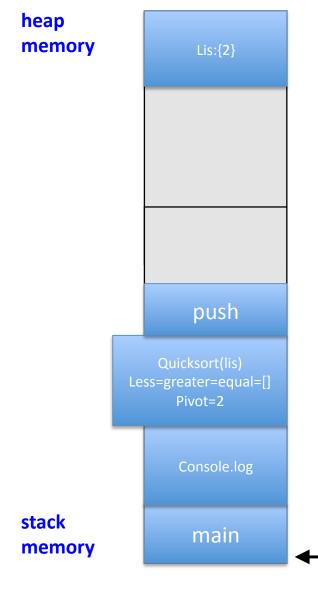
```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
    (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```

heap memory Console.log stack main memory

```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```



```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```



```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;)</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```

```
heap
memory
                   Quicksort([])
                Less=greater=[]
                  Equal=[2]
                   Pivot=2
                   Console.log
stack
                     main
memory
```

```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if (pivot<arra
     greater.
                  (array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```

#### memory Lis:{2} Array=[] Less=greater=[] Equal=[2] Pivot=2 Console.log stack main memory

heap

```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```

```
heap
memory
               Less=greater=[]
                  Equal=[2]
                  Pivot=2
                  Console.log
stack
                    main
memory
```

```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```

```
heap
memory
               Less=greater=[]
                  Equal=[2]
                  Pivot=2
                  Console.log
stack
                    main
memory
```

```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```



# heap memory Console.log stack main memory

```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```

memory stack main memory

heap

```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
   (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```

```
heap
memory
stack
```

memory

```
Function quicksort(array) {
If(array.length<=0) return[]</pre>
   var less=[];
   var greater=[];
   var equal=[]
   var pivot=array[0];
 for(var i=0;i<array.length;i+</pre>
   +)
 if(pivot>array[i])
      less.push(array[i]);
 if(pivot==array[i])
      equal.push(array[i]);
 if(pivot<array[i])</pre>
     greater.push(array[i]);
Return quicksort(less).concat
    (equal, quicksort (greater));
Var lis=new array(2);
Console.log(quicksort(lis));
```

#### Conclusion R1.3

- I did not need any modifications to the current stack and heap model to show how recursion works with javascript functions.
- Modifications to simplify the model would be greatly appreciated however, as this example used a lot of slide for such a simple case, The function works for much larger cases but the provided model is a bit too detailed for anything much larger than this.