Part 1

I will consider assignments to be one data movement and swaps to be two data movements. I will also consider index comparisons. I am ignoring other operations like indexing and math. The worst case for this version of quicksort is a reversed sorted array (but sorted is also pretty close).

code	data movements	comparisons	total
`def qsort(arr, beg, end):`			
`  if beg < end:`		1	1
`    pivot = arr[beg]`	1		1
`    i = beg+1`	1		1
`    j = end`	1		1
`    while i < j:`		1	1
`      while i<j and arr[i] <= pivot:`		2(n-1)	2n-2
`        i += 1`	n-1		n-1
`      while arr[j] >= pivot and i<j:`		2	2
`        j -= 1`	0		
`      if i<j:`		1	1
`        arr[i], arr[j] = arr[j], arr[i]`	0		
`      else: break`			
`    if arr[j] < pivot:`		1	1
`      arr[beg], arr[j] = arr[j], arr[beg]`	2		2
`    qsort(iterable, beg, j-1)`			
`    qsort(iterable, j+1, end)`		1	1
total:	n+4	2n+5	3n+9

Then, the function recurses. The second function call is the base case (1 comparison, as shown in the table). The first function call basically has n reduced by one each time, until n<1 (two base case recursive calls). That means that the complexity can be expressed as the following sum:

$$2 + \sum_{i=1}^{n} 3i + 9$$

That sum can be simplified into an expression that has O(n²) complexity:

$$= 9n + 2 + 3\sum_{i=1}^{n} i$$

$$= 9n + 2 + 3\frac{n(n+1)}{2}$$

$$= 9n + 2 + \frac{3}{2}n^2 + \frac{3}{2}n$$

$$= 1.5n^2 + 10.5n + 2$$

Part 2

`vector = [16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`

First qsort function on stack (beg = 0; end = 15):

- 16 is chosen as the pivot and `i` and `j` are assigned to the indices 1 and 15 respectively.
- `i` is incremented until `i=j` because `vector[i]` is never greater than the pivot.
- `j` is not decremented because `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are swapped.
- Two recursive calls are made, the second of which does nothing because it is the base case.

Current value of `vector` is [1, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 16]

Recursion 1 (beg = 0; end = 14):

- 1 is chosen as the pivot and `i` and `j` are assigned to the indices 1 and 14 respectively.
- `i` is not incremented because `vector[i]` is greater than the pivot.
- `j` is decremented until `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are not swapped because the pivot is less.
- Two recursive calls are made, the first of which does nothing because it is the base case.

Recursion 2 (beg = 1; end = 14):

- 15 is chosen as the pivot and `i` and `j` are assigned to the indices 2 and 14 respectively.
- `i` is incremented until `i=j` because `vector[i]` is never greater than the pivot.
- `j` is not decremented because `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are swapped.
- Two recursive calls are made, the second of which does nothing because it is the base case.

Current value of `vector` is [1, 2, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 15, 16]

Recursion 3 (beg = 1; end = 13):

- 2 is chosen as the pivot and `i` and `j` are assigned to the indices 2 and 13 respectively.
- `i` is not incremented because `vector[i]` is greater than the pivot.
- `j` is decremented until `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are not swapped because the pivot is less.
- Two recursive calls are made, the first of which does nothing because it is the base case.

Recursion 4 (beg = 2; end = 13):

- 14 is chosen as the pivot and `i` and `j` are assigned to the indices 3 and 13 respectively.
- `i` is incremented until `i=j` because `vector[i]` is never greater than the pivot.
- `j` is not decremented because `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are swapped.
- Two recursive calls are made, the second of which does nothing because it is the base case.

Current value of `vector` is [1, 2, 3, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 14, 15, 16]
Recursion 5 (beg = 2; end = 12):

- 3 is chosen as the pivot and `i` and `j` are assigned to the indices 3 and 12 respectively.
- `i` is not incremented because `vector[i]` is greater than the pivot.
- `j` is decremented until `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are not swapped because the pivot is less.
- Two recursive calls are made, the first of which does nothing because it is the base case.

Recursion 6 (beg = 3; end = 12):

- 13 is chosen as the pivot and `i` and `j` are assigned to the indices 4 and 12 respectively.
- `i` is incremented until `i=j` because `vector[i]` is never greater than the pivot.
- `j` is not decremented because `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are swapped.
- Two recursive calls are made, the second of which does nothing because it is the base case.

Current value of `vector` is [1, 2, 3, 4, 12, 11, 10, 9, 8, 7, 6, 5, 13, 14, 15, 16]
Recursion 7 (beg = 3; end = 11):

- 4 is chosen as the pivot and `i` and `j` are assigned to the indices 4 and 11 respectively.
- `i` is not incremented because `vector[i]` is greater than the pivot.
- `j` is decremented until `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are not swapped because the pivot is less.
- Two recursive calls are made, the first of which does nothing because it is the base case.

Recursion 8 (beg = 4; end = 11):

- 12 is chosen as the pivot and `i` and `j` are assigned to the indices 5 and 11 respectively.
- `i` is incremented until `i=j` because `vector[i]` is never greater than the pivot.
- `j` is not decremented because `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are swapped.
- Two recursive calls are made, the second of which does nothing because it is the base case.

Current value of `vector` is [1, 2, 3, 4, 5, 11, 10, 9, 8, 7, 6, 12, 13, 14, 15, 16]
Recursion 9 (beg = 4; end = 10):

- 5 is chosen as the pivot and `i` and `j` are assigned to the indices 5 and 10 respectively.
- `i` is not incremented because `vector[i]` is greater than the pivot.
- `j` is decremented until `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are not swapped because the pivot is less.
- Two recursive calls are made, the first of which does nothing because it is the base case.

Recursion 10 (beg = 5; end = 10):

- 11 is chosen as the pivot and `i` and `j` are assigned to the indices 6 and 10 respectively.
- `i` is incremented until `i=j` because `vector[i]` is never greater than the pivot.
- `j` is not decremented because `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are swapped.
- Two recursive calls are made, the second of which does nothing because it is the base case.

Current value of `vector` is [1, 2, 3, 4, 5, 6, 10, 9, 8, 7, 11, 12, 13, 14, 15, 16]
Recursion 11 (beg = 5; end = 9):

- 6 is chosen as the pivot and `i` and `j` are assigned to the indices 6 and 9 respectively.
- `i` is not incremented because `vector[i]` is greater than the pivot.
- `j` is decremented until `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are not swapped because the pivot is less.
- Two recursive calls are made, the first of which does nothing because it is the base case.

Recursion 12 (beg = 6; end = 9):

- 10 is chosen as the pivot and `i` and `j` are assigned to the indices 7 and 9 respectively.
- `i` is incremented until `i=j` because `vector[i]` is never greater than the pivot.
- `j` is not decremented because `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are swapped.
- Two recursive calls are made, the second of which does nothing because it is the base case.

Current value of `vector` is [1, 2, 3, 4, 5, 6, 7, 9, 8, 10, 11, 12, 13, 14, 15, 16]
Recursion 13 (beg = 6; end = 8):

- 7 is chosen as the pivot and `i` and `j` are assigned to the indices 7 and 8 respectively.
- `i` is not incremented because `vector[i]` is greater than the pivot.
- `j` is decremented until `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are not swapped because the pivot is less.
- Two recursive calls are made, the first of which does nothing because it is the base case.

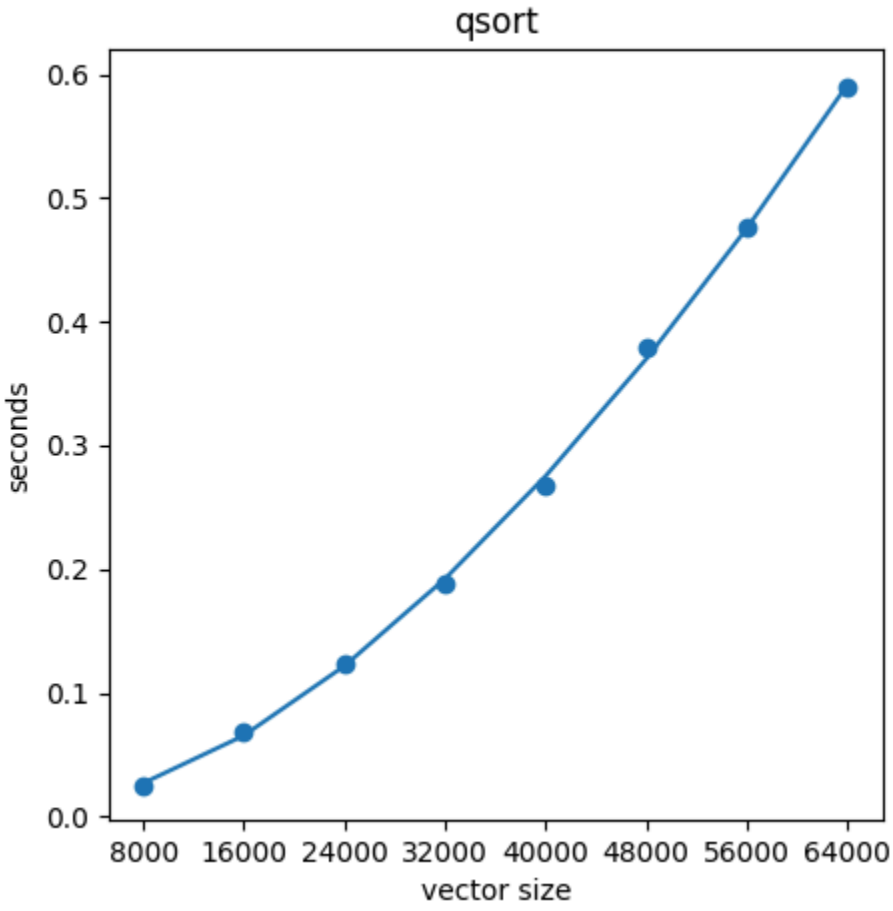
Recursion 14 (beg = 7; end = 8):

- 9 is chosen as the pivot and `i` and `j` are both assigned to the index 8.
- `i` is not incremented because `i=j`.
- `j` is not decremented because `i=j`.
- `vector[i]` and `vector[j]` are not swapped because `i=j`.
- `vector[j]` and the pivot are swapped.
- Two recursive calls are made, both of which do nothing because they are base cases.

The stack is collapsed and the final value of `vector` is
`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]`

Part 4

I tested various functions to fit my data to and the one that worked best was of the form $a \times n^2 + b \times n \log(n)$. This does match my complexity analysis in that the function is quadratic, but I did not predict a log-linear term. This was one of the graphs that I got:



The values of a and b for this graph were about 9.61e-11 and 1.96e-07.