Programming Assignment 2
Digital Library of Arendelle

Breno Poggiali de Sousa
brenopoggiali@ufmg.br

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

1 Introduction

The Arendelle's Digital Library is a project developed by the Plan of Democratization of the Education Access (PDEA). This project aims to guarantee all the Arendelle people the right of information access from the Internet. The problem proposed by Elsa, Arendelle's Queen, was to develop an efficient sort algorithm that could be able to sort the huge Arendelle's Digital Library. Since it is a massive amount of books, we need to find an effective quicksort variation that could be able to sort the book fast and, in this paper, we will try to compare many quick sort variations to give the final report to Elsa about which is the most effective one.

2 Implementation

All the quicksort implementations were developed in C++ and compiled by G++ of GNU Compiler Collection

2.1 Data Structure

Most of the QuickSorts were designed using a recursive implementation, except for the last one that we used a **Stack** to simulate the recursive implementation. A recursive implementation is when some function calls itself and it always needs to have a stop condition. After we achieve the stop condition, the last function starts to return to its predecessor and so on. Just like pushing elements to a Stack and then popping it until it gets empty.

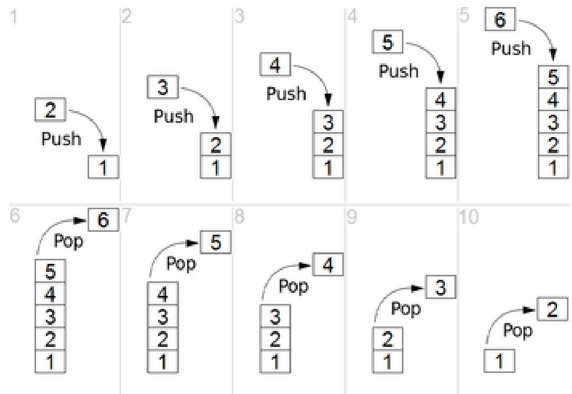
So this is what is happening inside a stack:

In this example of an implementation of a Stack of length equals to 6, we first push all the 6 elements and then, when the stop condition is reached, we start to pop each element out, until the end of the stack.

Ok, but which are these elements that we are pushing and popping out of the stack? These elements, in our case, were the left and right edges of our vector. Considering that our vector represents each of our books.

Since we are using quicksort implementations, we are always dividing our vector in half and, these elements that we are pushing and popping, are the left and right corners of the divided vector.

We will explain more about QuickSort implementations latter on this paper.



Lifo_stack.png

2.2 Classes

Keeping in mind the idea of letting our program more modular, we tried to separate the whole program in some classes. We divided it into two main classes besides our `main.cpp` These two classes are:

2.2.1 Array

In our array class, we tried to develop all of our QuickSort implementations and all our array handling. We initialize this class passing only the size of the array and it auto creates an ascending ordered array and set `moves = 0` and `comparisons = 0`. If we are trying to use and descending ordered or a shuffled array, we only call `desc_array()` or `shuffle_array()` respectively.

Then, when we call any quicksort but non recursive quicksort, we call the `quicksort()` function passing the following parameters: `(unsigned int* A, int esq, int dir, int type, int k)`, where `A` is our Array of elements, `esq` and `dir` are the left and right edges of our array that we are looking for, `type` is the type of our pivot that we are using and k is the percentage of the vector that we will use in case of doing the Insertion QuickSort (look that if that is not the case, we set k=0 and we will never satisfy the if that call the insertion sort).

In the case that we are looking for the non recursive quicksort, we call the `quicksort_non_recursive` that acts pretty similar as the quicksort function that we saw above, **except** that we only need to pass `type` and `k`, since, in this case, we will only have one case (pivot = the middle element and k = 0 - since we are not doing a non recursive insertion sort). And, of course, rather than calling itself, this function has a while that only stops when the stack that it created is empty

The other functions do pretty much what their name is saying:

- `which_pivot()` returns the pivot of the quicksort case that we are trying to do.
- `print_array()` print the array (we use it when -p is passed)
- `get_moves()` and `get_comparisons()` returns the number of moves and comparisons that was calculated during the quicksorts.
- `get_median()` gets the median of the array (we use this when we are getting the median of the sorted time array). See that, when we call this function, we have executed 20 examples and insert the time at the `median_time` array. After this, we sort the `median_time` array and then, we call this function to get the median of this array.

And, of course, all the other functions: `get_qc()`, `get_qm3()`, `get_qpe()`, `get_qi()`, `get_qnr()` calls the quicksort functions with the right parameters.

2.2.2 Stack

The class Stack is much more simple than Array class. It has the minimal functions necessary to create a stack and allow us to use it. Popping and Pushing elements of it.

Our constructor here sets the top of the Stack to 0 and creates a stack of the size that was passed (in our case, the size of elements in the array).

Then we have the `push()` function that pushes the edges to the Stack (the left and the right edges). And sets `top = top + 1`

And the `pop()` function that return theses edges. And sets `top = top - 1`

And last but not least, we have the `is_empty()` function that returns a boolean of whether the Stack is empty or not.

2.3 Quick Sorts

2.3.1 Classic Quick Sort

As we can see from the graphs above, this was the quicksort with the best efficiency and it definitely is the one that Elza is looking for. It uses the central element of each array as its pivot.

2.3.2 Median of 3 Quick Sort

This quicksort uses the median between the first, the middle and the last elements as its pivot. But as we can see from the results, we don't have enough efficiency gain and it is a little slower than the classic one.

2.3.3 First element Quick Sort

Wow.. this is one of the most painful quicksorts in terms of computational cost. Especially when we look at the OrdC or the OrdD times, we can see here that it can take over to 5 minutes ordering one ascending vector of 500,000 elements. Here, of course, we use the first element of an array as the pivot.

2.3.4 Insertion Quick Sort

Here we have sort of a hybrid implementation. Since we use quick sort's implementation until the vector has `k%` elements and then, we call an insertion sort algorithm, we have a little bit of lower efficiency when compared to the classic one. Here we use the middle element as the pivot.

One interesting fact here is that, when we get bigger `k`'s, we have a considerative lower of efficiency. Which take us to the conclusion that the quicksort algorithm is much more efficient than the insertion sort one.

2.3.5 Non-recursive Quick Sort

Here we have an implementation that tries to simulate the Classic Quick Sort and that's because we have such a good result. Therefore, since we are implementing a Stack by hand, our efficiency gets a little bit lower than the Classic one.

3 How to use

To execute the program, you need to compile it first.

3.1 Compile

Since we have a Makefile, to compile the file, you only need to open its main repository and execute the following command:

```
make
```

3.2 Execution

To execute you have two options. The first one is to execute all the tests. In this case, you only need to execute:

```
./main
```

and all the tests will be executed.

If you prefer to test each case separelaty, you can run:

```
./main <quickSort> <type> <size>
```

For example:

```
./main QNR OrdC 250000
```

will execute the non-recursive quicksort implementation of an ordered array of length 250000.

If you want to test print the arrays that were used in your test, you just need to add `-p` at the end.

For example:

```
./main QC Ale 10 -p
```

will execute the common quicksort implementation in a shuffled array of length 10 and print all the 20 arrays that were used in the tests.

3.3 Output

The output will contain the quicksort variation, type, size, the average number of comparisons, the average number of moves and the median of the time (**in microseconds**) of all the 20 tests and, if it contains the `-p` parameter, it will also have all the vectors used in each case.

Example:

```
[Input] ./main QPE OrdC 10
[Output] QPE Ale 10 45 13 2
```

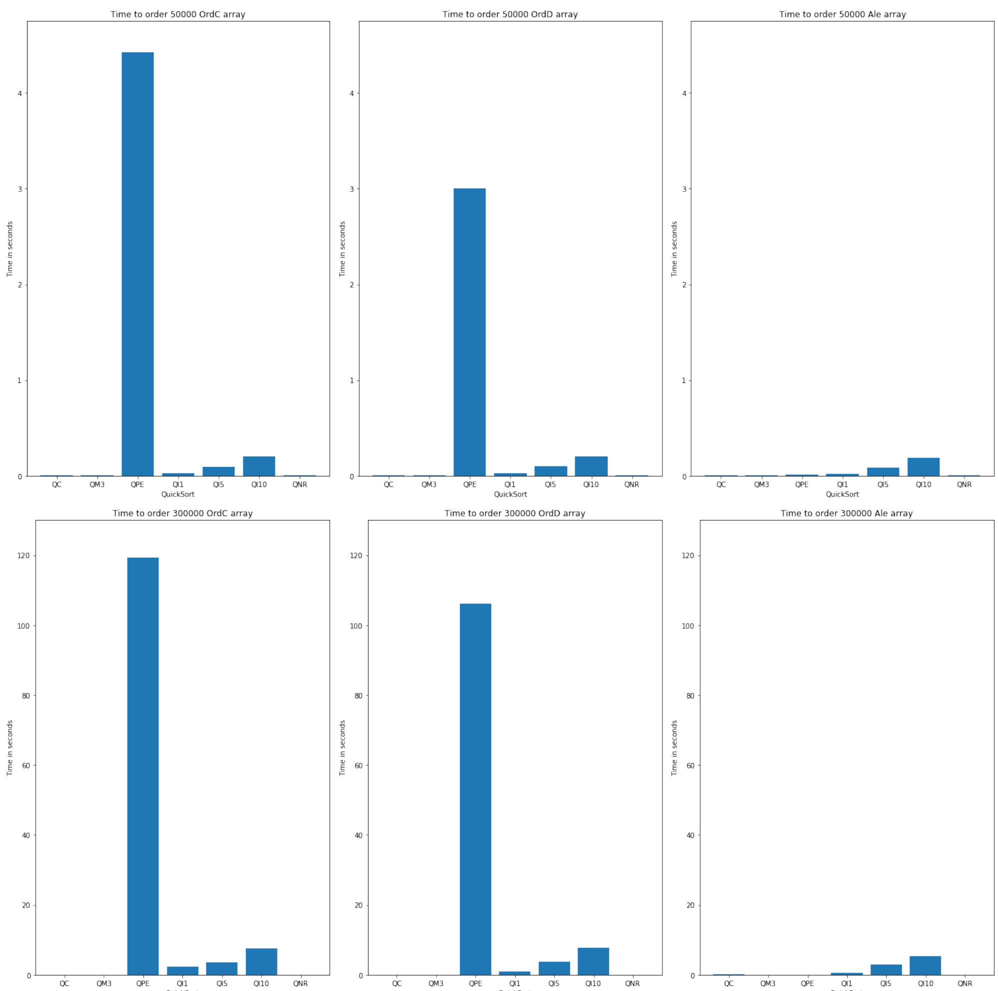
or, with `-p`

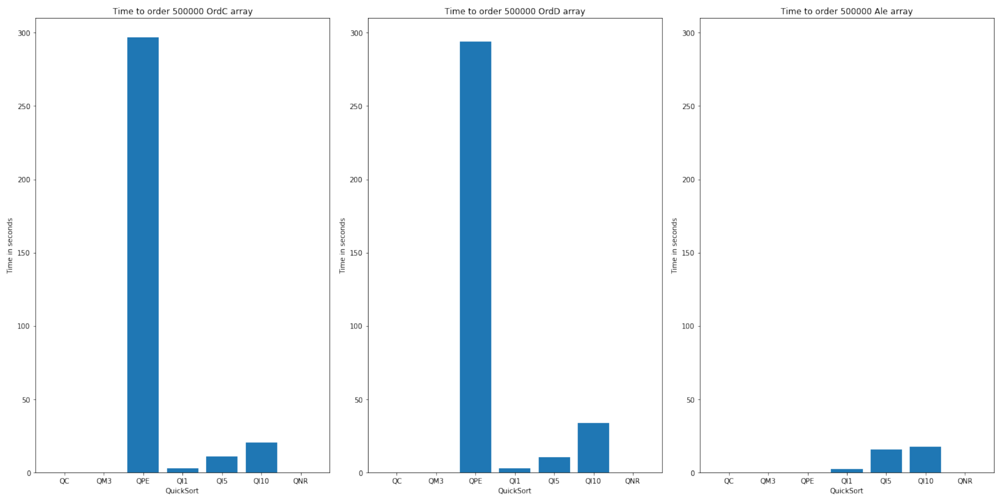
```
[Input] ./main QM3 Ale 10 -p
[Output] QM3 Ale 10 42 13 2
6 6 1 1 7 5 7 1 5 9
1 9 6 2 5 5 0 5 7 9
4 3 2 6 7 6 7 6 4 0
3 1 6 1 4 6 3 2 3 9
6 2 8 1 7 6 1 0 9 6
.
.
.
```

4 Speed analysis

In this session, we will compare all quicksorts and see which one is the best for Elsa to use.

4.1 Timing





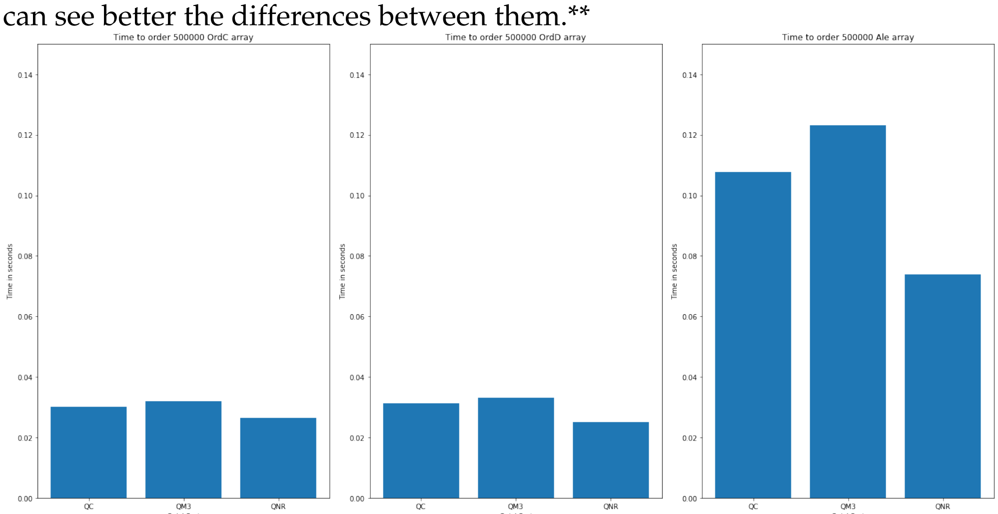
4.1.1 The worst timing scenario

Looking at the graphs above, we just saw that the QPE (First element QuickSort) is the **slowest** one. The worst scenario in this QS is the OrdC one, but we can see here that, when we get bigger vectors, the timing of OrdC and OrdD gets more similar.

It is interesting to look here as well that, when we use QPE in a shuffled array (Ale), our timing is actually pretty ok, compared to the ordered ones. This occurs, because, since we are using the first element as the pivot, in the worst case (ordered array), rather than go until the half of the array and then split, O(n log n), the array is only split when it goes through all the array. So we have an O(n^2).

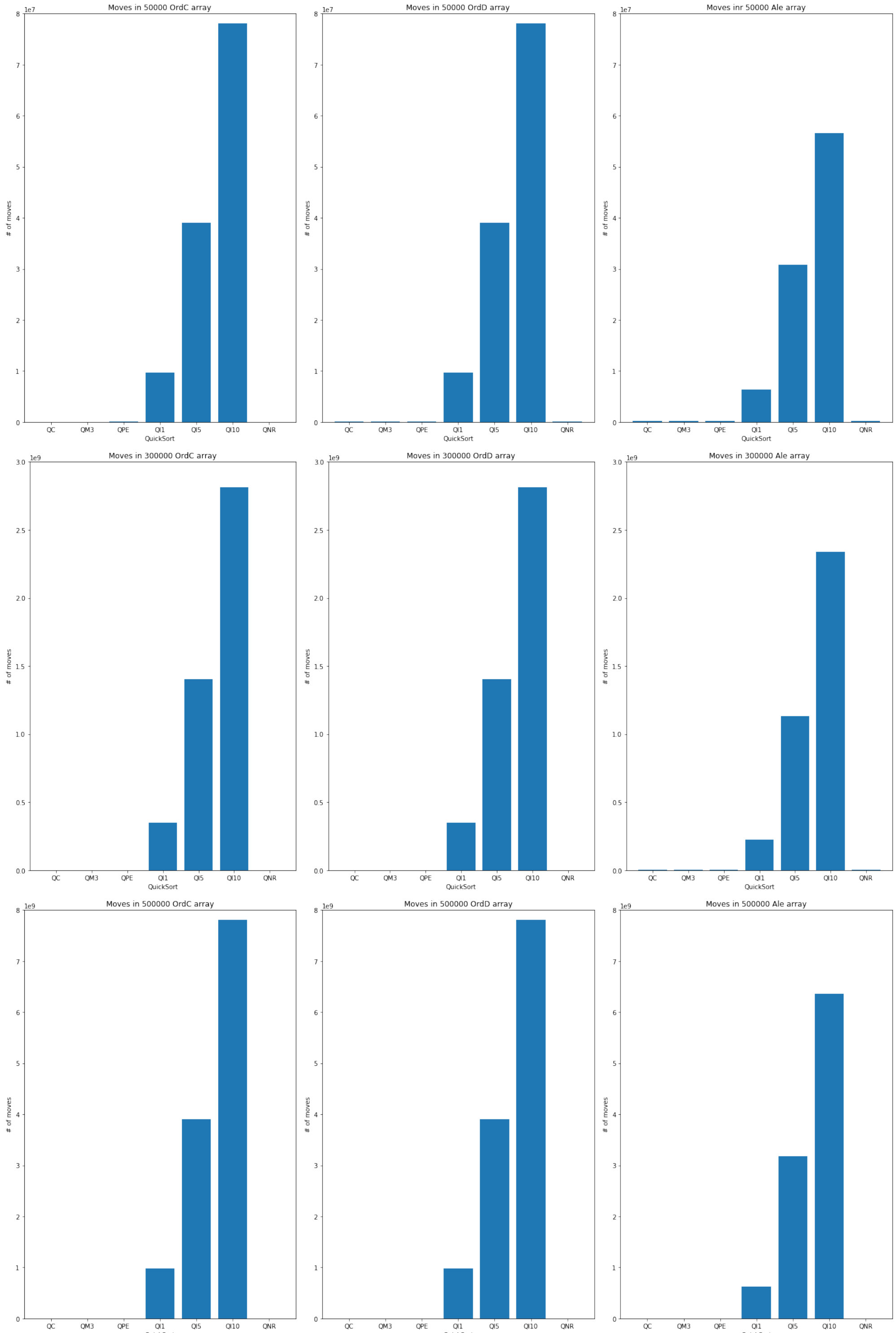
4.1.2 The best timing scenario

According to our graphs, the quicksorts QC, QM3 and QNR are pretty similar. So we will plot only these 3 to compare them. We will plot only the 500000 length array because is the largest and we can see better the differences between them.**



Now we can see that the fastest timing is the QNR (Non-recursive QuickSort) one. This can be explained, maybe, because the recursive function isn't doing only what it should do - since it is only one function for all the recursive quicksort cases.

4.2 Moves



4.2.1 The worst moving scenario

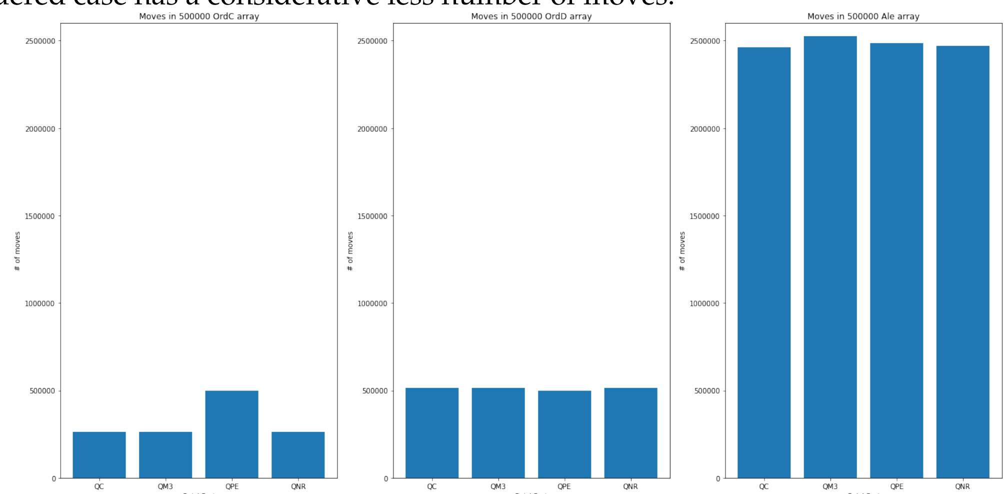
Looking at the graphs above, we just saw that the QI (Insertion Quicksort) is the one with most moves. This was the expected result since the insertion quicksort uses the insertion implementation and it is the most costly in terms of movements. The QI10 is higher, of course, because the QI10 uses more Insertion than the QI5 and QI1.

Here we can see that the ordered ones are the most costly as well, because, in these cases, the worst scenario happens more times than the random one.

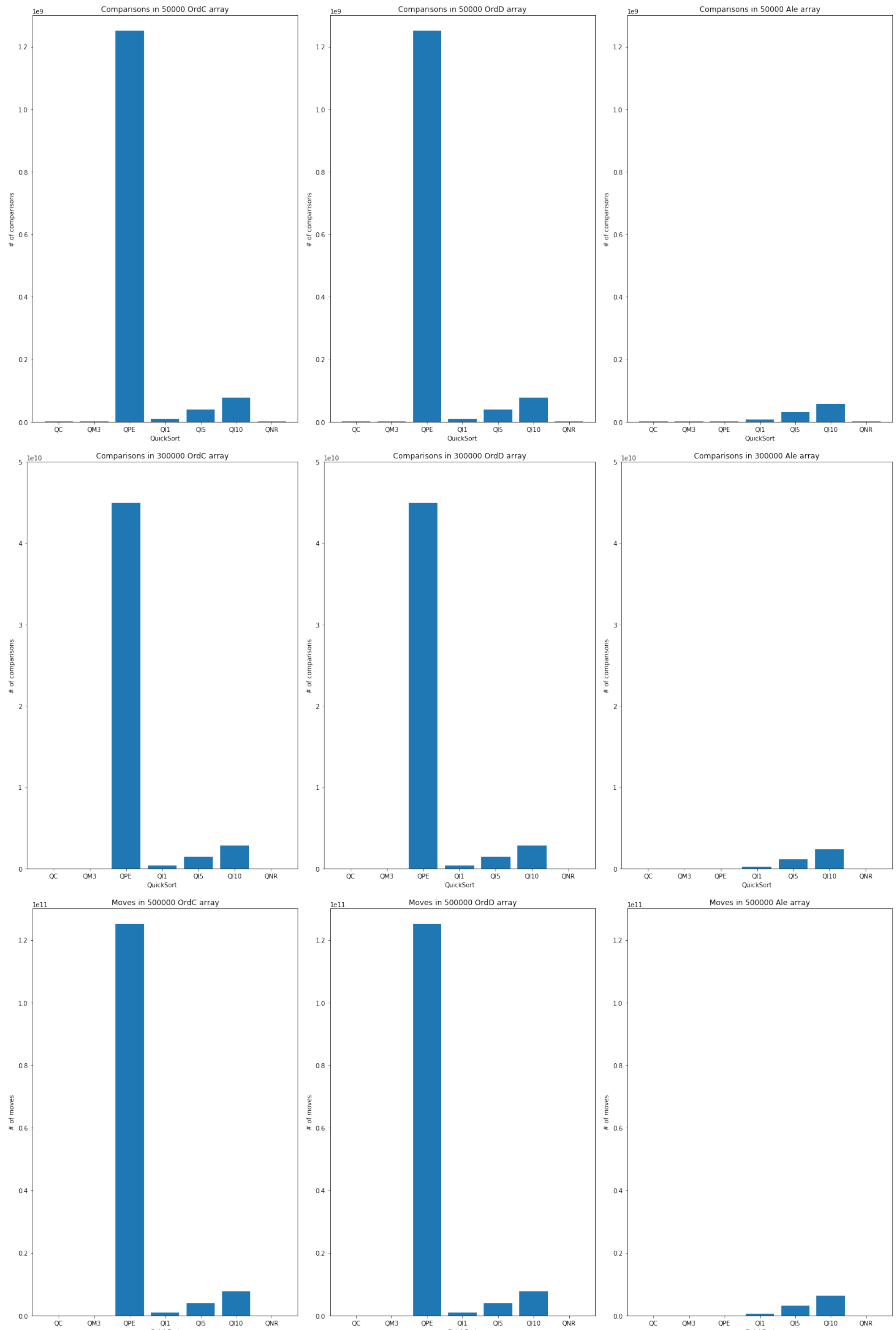
4.2.2 The best moving scenario

According to our graphs, the quicksorts QC, QM3, QPE and QNR are pretty similar. So we will plot only these 4 to compare them. We will plot only the 500000 length array because is the largest and we can see better the differences between them.

Now, we can see that the count of movements of these 4 is pretty similar. And, because of this, we can't guarantee that one is much better than the other. Anyway, we can see here, that now, the ordered case has a considerative less number of moves.



4.3 Comparison



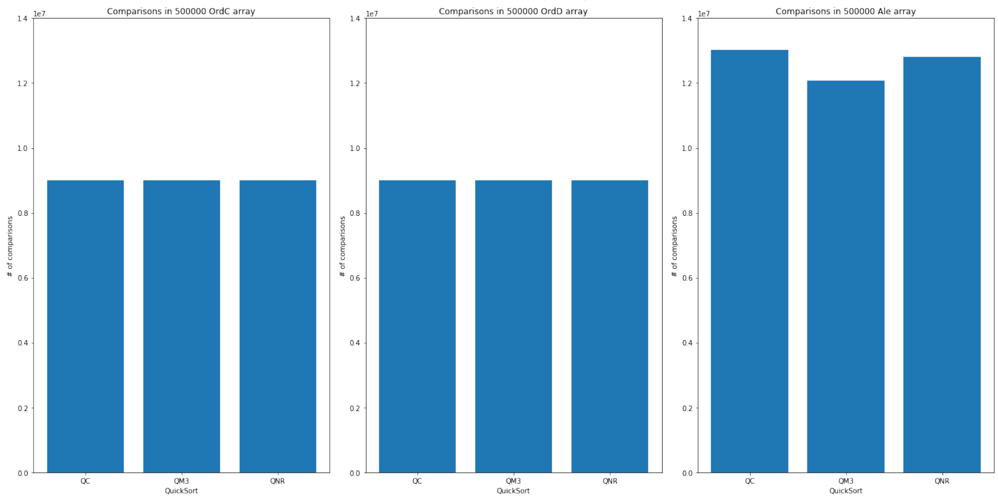
4.3.1 The worst comparison scenario

Here we have exactly what we expected. Since the QPE is using the first element of each vector as the pivot, it will be the one, for sure with most comparisons in the worst case (ordered arrays). In the shuffled one, we have the insertion quicksort as the worst case, because, in the insertion sort, we are always comparing to see when to stop, meanwhile in the QPE, since we have a shuffled array, we will find faster one case the finish the for.

4.3.2 The best comparison scenario

According to our graphs, the quicksorts QC, QM3 and QNR are pretty similar. So we will plot only these 3 to compare them. We will plot only the 500000 length array because is the largest and we can see better the differences between them.

Here we can see that in the ordered array they are pretty similar (since the QM3 will always have its pivot as the array in the middle element). But we can see that the QM3, in the shuffled one has a slightly smaller number of comparisons. This can be explained by the fact that since it gets the median between 3 numbers, we always get the one that is more in the middle.



5 Conclusion

To finish this paper and to give a decent solution to Elsa, we can confirm that the **common quick-Sort** is definitely the better solution for the queen of Arendelle's program. And, just to be clear, when we refer to the common quickSort, we are referring to the QC and the QNR, that uses the same implementation. This because they both have a very good and similar performance when dealing with big numbers in all the scenarios.

Elsa may prefer to use the QC because it's implementation is easier since she doesn't need to create a Stack to simulate the recursive function. And also, when developing a function to deal only with the QC case, she might get a little performance gain.

6 References

Ziviani, N. (2006). *Projetos de Algoritmos com Implementações em PASCAL e C: Capítulo 4: Ordenação*. Editora Cengage.

A Results from my own machine: (used in the plots)

```
QC Ale 50000 1061764 208110 6613
QC Ale 100000 2257713 439520 13727
QC Ale 150000 3534231 679694 29580
QC Ale 200000 4763714 926445 43388
QC Ale 250000 6135936 1173376 59510
QC Ale 300000 7486617 1424746 61941
QC Ale 350000 8895041 1678393 81263
QC Ale 400000 10185034 1940150 76413
QC Ale 450000 11539280 2201062 98855
QC Ale 500000 13011936 2462290 107708
QC OrdC 50000 750015 32767 2673
QC OrdC 100000 1600016 65535 8447
QC OrdC 150000 2456803 84464 8232
QC OrdC 200000 3400017 131071 12601
QC OrdC 250000 4250017 131071 15120
QC OrdC 300000 5213588 168928 17518
QC OrdC 350000 6213588 218928 22737
QC OrdC 400000 7200018 262143 24821
QC OrdC 450000 8100018 262143 29591
QC OrdC 500000 9000018 262143 30017
QC OrdD 50000 750028 57766 2685
QC OrdD 100000 1600030 115534 6003
QC OrdD 150000 2456820 159464 8621
QC OrdD 200000 3400032 231070 12099
QC OrdD 250000 4250032 256070 15147
QC OrdD 300000 5213606 318928 18147
QC OrdD 350000 6213606 393928 24555
QC OrdD 400000 7200034 462142 26481
QC OrdD 450000 8100034 487142 31185
QC OrdD 500000 9000034 512142 31254
QM3 Ale 50000 995656 213407 8974
QM3 Ale 100000 2107078 450502 17558
QM3 Ale 150000 3272236 696788 27081
QM3 Ale 200000 4467416 948803 37274
QM3 Ale 250000 5687475 1204535 46424
QM3 Ale 300000 6882475 1465419 56529
QM3 Ale 350000 8131051 1729295 69968
QM3 Ale 400000 9458076 1990577 79383
QM3 Ale 450000 10716501 2256810 86380
QM3 Ale 500000 12063768 2525702 123199
QM3 OrdC 50000 750015 32767 2663
QM3 OrdC 100000 1600016 65535 6677
QM3 OrdC 150000 2456803 84464 11654
QM3 OrdC 200000 3400017 131071 12559
QM3 OrdC 250000 4250017 131071 15325
QM3 OrdC 300000 5213588 168928 19850
```

```
QM3 OrdC 350000 6213588 218928 23044
QM3 OrdC 400000 7200018 262143 26071
QM3 OrdC 450000 8100018 262143 28330
QM3 OrdC 500000 9000018 262143 32064
QM3 OrdD 50000 750028 57766 3003
QM3 OrdD 100000 1600030 115534 7161
QM3 OrdD 150000 2456820 159464 10913
QM3 OrdD 200000 3400032 231070 13681
QM3 OrdD 250000 4250032 256070 16145
QM3 OrdD 300000 5213606 318928 19068
QM3 OrdD 350000 6213606 393928 25621
QM3 OrdD 400000 7200034 462142 31589
QM3 OrdD 450000 8100034 487142 31947
QM3 OrdD 500000 9000034 512142 33198
QPE Ale 50000 1214036 210366 9563
QPE Ale 100000 2585203 444198 18074
QPE Ale 150000 4019063 687680 29607
QPE Ale 200000 5587339 932401 39243
QPE Ale 250000 6981621 1186908 51101
QPE Ale 300000 8529376 1439126 59708
QPE Ale 350000 10074289 1697778 68256
QPE Ale 400000 11579641 1962326 81491
QPE Ale 450000 13343069 2221223 95672
QPE Ale 500000 14823250 2486175 118951
QPE OrdC 50000 1250074998 49999 4426739
QPE OrdC 100000 5000149998 99999 18700636
QPE OrdC 150000 11250224998 149999 38786909
QPE OrdC 200000 20000299998 199999 52134110
QPE OrdC 250000 31250374998 249999 120417782
QPE OrdC 300000 45000449998 299999 119301036
QPE OrdC 350000 61250524998 349999 153046754
QPE OrdC 400000 80000599998 399999 259352929
QPE OrdC 450000 101250674998 449999 240067932
QPE OrdC 500000 125000749998 499999 297011501
QPE OrdD 50000 1250099996 49999 3002425
QPE OrdD 100000 5000199996 99999 11941522
QPE OrdD 150000 11250299996 149999 26779448
QPE OrdD 200000 20000399996 199999 47206506
QPE OrdD 250000 31250499996 249999 73699318
QPE OrdD 300000 45000599996 299999 106131453
QPE OrdD 350000 61250699996 349999 144594569
QPE OrdD 400000 80000799996 399999 188683827
QPE OrdD 450000 101250899996 449999 239163078
QPE OrdD 500000 125000999996 499999 294031909
QI1 Ale 50000 6766551 6407681 22801
QI1 Ale 100000 25989873 25277930 73504
QI1 Ale 150000 57445334 56377200 158007
QI1 Ale 200000 103746915 102299688 283377
```

```
QI1 Ale 250000 159870561 158095246 435030
QI1 Ale 300000 228570388 226408641 620559
QI1 Ale 350000 317392380 314922977 959166
QI1 Ale 400000 398088527 395228584 1074295
QI1 Ale 450000 513388505 510100410 1985015
QI1 Ale 500000 625717504 622134147 2160949
QI1 OrdC 50000 10041164 9691284 25556
QI1 OrdC 100000 39613427 38913547 108196
QI1 OrdC 150000 88716937 87667057 238326
QI1 OrdC 200000 157351712 155951832 512829
QI1 OrdC 250000 245517719 243767839 706614
QI1 OrdC 300000 353214989 351115109 2263584
QI1 OrdC 350000 480443508 477993628 1443056
QI1 OrdC 400000 627203259 624403379 1614785
QI1 OrdC 450000 793494289 790344409 2051418
QI1 OrdC 500000 979316552 975816672 2524155
QI1 OrdD 50000 10041558 9716672 29402
QI1 OrdD 100000 39614212 38964326 133025
QI1 OrdD 150000 88718112 87743226 266413
QI1 OrdD 200000 157353278 156053392 524095
QI1 OrdD 250000 245519676 243894790 671463
QI1 OrdD 300000 353217336 351267450 893259
QI1 OrdD 350000 480446246 478171360 1708751
QI1 OrdD 400000 627206388 624606502 1611973
QI1 OrdD 450000 793497808 790572922 2619654
QI1 OrdD 500000 979320462 976070576 2961240
QI5 Ale 50000 31012535 30781419 86401
QI5 Ale 100000 124376064 123911427 483951
QI5 Ale 150000 288963405 288277778 982244
QI5 Ale 200000 516339670 515396562 1487315
QI5 Ale 250000 770501360 769334561 2637159
QI5 Ale 300000 1133272117 1131818958 3047108
QI5 Ale 350000 1574892788 1573288208 3948376
QI5 Ale 400000 2010752114 2008877341 6760852
QI5 Ale 450000 2561873907 2559732067 9133287
QI5 Ale 500000 3178563385 3176234789 15544073
QI5 OrdC 50000 39239102 38989128 96723
QI5 OrdC 100000 156603161 156103187 385683
QI5 OrdC 150000 352092227 351342253 925021
QI5 OrdC 200000 625706286 624706312 1601094
QI5 OrdC 250000 977445352 976195378 3499339
QI5 OrdC 300000 1407309411 1405809437 3551540
QI5 OrdC 350000 1915298477 1913548503 5691615
QI5 OrdC 400000 2501412536 2499412562 6547606
QI5 OrdC 450000 3165651602 3163401628 8564268
QI5 OrdC 500000 3908015661 3905515687 10786112
QI5 OrdD 50000 39240666 39015688 99773
QI5 OrdD 100000 156606288 156156310 408366
```

```
QI5 OrdD 150000 352096916 351421938 880309
QI5 OrdD 200000 625712538 624812560 2221833
QI5 OrdD 250000 977453166 976328188 2634931
QI5 OrdD 300000 1407318788 1405968810 3742992
QI5 OrdD 350000 1915309416 1913734438 5247604
QI5 OrdD 400000 2501425038 2499625060 6706089
QI5 OrdD 450000 3165665666 3163640688 9240492
QI5 OrdD 500000 3908031288 3905781310 10540792
QI10 Ale 50000 56739711 56558990 185850
QI10 Ale 100000 251235195 250873270 620260
QI10 Ale 150000 565628949 565073493 1307171
QI10 Ale 200000 1013739586 1012991589 2684356
QI10 Ale 250000 1489374809 1488402856 3974389
QI10 Ale 300000 2342022405 2340924749 5424809
QI10 Ale 350000 3127559052 3126304624 7800749
QI10 Ale 400000 3967550051 3966121377 13527398
QI10 Ale 450000 5413909180 5412368317 15236742
QI10 Ale 500000 6362274785 6360480631 17534330
QI10 OrdC 50000 78253144 78053155 199708
QI10 OrdC 100000 312756269 312356280 786394
QI10 OrdC 150000 703509394 702909405 1766748
QI10 OrdC 200000 1250512519 1249712530 3167847
QI10 OrdC 250000 1953765644 1952765655 5867365
QI10 OrdC 300000 2813268769 2812068780 7556404
QI10 OrdC 350000 3829021894 3827621905 9725355
QI10 OrdC 400000 5001025019 4999425030 12790424
QI10 OrdC 450000 6329278144 6327478155 18362113
QI10 OrdC 500000 7813781269 7811781280 20431878
QI10 OrdD 50000 78256270 78081278 199634
QI10 OrdD 100000 312762520 312412528 793683
QI10 OrdD 150000 703518770 702993778 2207009
QI10 OrdD 200000 1250525020 1249825028 3279818
QI10 OrdD 250000 1953781270 1952906278 5178539
QI10 OrdD 300000 2813287520 2812237528 7838011
QI10 OrdD 350000 3829043770 3827818778 10320523
QI10 OrdD 400000 5001050020 4999650028 13962457
QI10 OrdD 450000 6329306270 6327731278 17790187
QI10 OrdD 500000 7813812520 7812062528 33734904
QNR Ale 50000 1062445 207995 6257
QNR Ale 100000 2241573 440117 13686
QNR Ale 150000 3506285 679107 21535
QNR Ale 200000 4760357 925580 28104
QNR Ale 250000 6090828 1175892 36759
QNR Ale 300000 7492738 1424845 44159
QNR Ale 350000 8786772 1682145 51932
QNR Ale 400000 10089666 1944470 61844
QNR Ale 450000 11515835 2203357 70961
QNR Ale 500000 12806946 2468465 73821
```

```
QNR OrdC 50000 750015 32767 2547
QNR OrdC 100000 1600016 65535 5031
QNR OrdC 150000 2456803 84464 7776
QNR OrdC 200000 3400017 131071 9813
QNR OrdC 250000 4250017 131071 14773
QNR OrdC 300000 5213588 168928 17031
QNR OrdC 350000 6213588 218928 17637
QNR OrdC 400000 7200018 262143 20244
QNR OrdC 450000 8100018 262143 23479
QNR OrdC 500000 9000018 262143 26507
QNR OrdD 50000 750028 57766 2466
QNR OrdD 100000 1600030 115534 4975
QNR OrdD 150000 2456820 159464 9499
QNR OrdD 200000 3400032 231070 12145
QNR OrdD 250000 4250032 256070 15357
QNR OrdD 300000 5213606 318928 15170
QNR OrdD 350000 6213606 393928 17676
QNR OrdD 400000 7200034 462142 20775
QNR OrdD 450000 8100034 487142 21376
QNR OrdD 500000 9000034 512142 25015
```

B Results from Google Cloud Platform Machine:

```
QC Ale 50000 1058884 208268 11124
QC Ale 100000 2264272 439054 47727
QC Ale 150000 3495496 679794 36275
QC Ale 200000 4765539 926320 49501
QC Ale 250000 6170820 1173713 68167
QC Ale 300000 7443548 1427609 113223
QC Ale 350000 8760440 1684891 112348
QC Ale 400000 10238884 1940129 150155
QC Ale 450000 11506687 2202850 159160
QC Ale 500000 12931712 2466530 205783
QC OrdC 50000 750015 32767 3509
QC OrdC 100000 1600016 65535 7371
QC OrdC 150000 2456803 84464 10894
QC OrdC 200000 3400017 131071 27441
QC OrdC 250000 4250017 131071 18558
QC OrdC 300000 5213588 168928 23041
QC OrdC 350000 6213588 218928 51801
QC OrdC 400000 7200018 262143 32443
QC OrdC 450000 8100018 262143 59535
QC OrdC 500000 9000018 262143 38979
QC OrdD 50000 750028 57766 11673
QC OrdD 100000 1600030 115534 7750
QC OrdD 150000 2456820 159464 11455
QC OrdD 200000 3400032 231070 16167
QC OrdD 250000 4250032 256070 19425
QC OrdD 300000 5213606 318928 23959
QC OrdD 350000 6213606 393928 28961
QC OrdD 400000 7200034 462142 33813
QC OrdD 450000 8100034 487142 36985
QC OrdD 500000 9000034 512142 40713
QM3 Ale 50000 994740 213620 17595
QM3 Ale 100000 2098497 451310 24379
QM3 Ale 150000 3259612 697576 49760
QM3 Ale 200000 4482356 948194 51219
QM3 Ale 250000 5666558 1205783 64835
QM3 Ale 300000 6973105 1462783 104890
QM3 Ale 350000 8184182 1726671 92973
QM3 Ale 400000 9424603 1993031 107250
QM3 Ale 450000 10748545 2256358 121485
QM3 Ale 500000 12034901 2526108 210068
QM3 OrdC 50000 750015 32767 9768
QM3 OrdC 100000 1600016 65535 13911
QM3 OrdC 150000 2456803 84464 23668
QM3 OrdC 200000 3400017 131071 16564
QM3 OrdC 250000 4250017 131071 19587
QM3 OrdC 300000 5213588 168928 24571
```

```
QM3 OrdC 350000 6213588 218928 29452
QM3 OrdC 400000 7200018 262143 34539
QM3 OrdC 450000 8100018 262143 55908
QM3 OrdC 500000 9000018 262143 65338
QM3 OrdD 50000 750028 57766 9931
QM3 OrdD 100000 1600030 115534 8271
QM3 OrdD 150000 2456820 159464 12230
QM3 OrdD 200000 3400032 231070 17223
QM3 OrdD 250000 4250032 256070 20539
QM3 OrdD 300000 5213606 318928 25400
QM3 OrdD 350000 6213606 393928 33454
QM3 OrdD 400000 7200034 462142 35944
QM3 OrdD 450000 8100034 487142 39251
QM3 OrdD 500000 9000034 512142 43006
QPE Ale 50000 1212152 210340 11736
QPE Ale 100000 2576453 444131 48603
QPE Ale 150000 4020202 686048 38289
QPE Ale 200000 5519012 933154 51722
QPE Ale 250000 7047830 1186190 66711
QPE Ale 300000 8521237 1441777 79772
QPE Ale 350000 10161614 1697008 139342
QPE Ale 400000 11758527 1956619 153180
QPE Ale 450000 13107618 2228501 145332
QPE Ale 500000 14818197 2486907 204289
QPE OrdC 50000 1250074998 49999 6319013
QPE OrdC 100000 5000149998 99999 23289690
QPE OrdC 150000 11250224998 149999 41245284
QPE OrdC 200000 20000299998 199999 73652630
QPE OrdC 250000 31250374998 249999 115176223
QPE OrdC 300000 45000449998 299999 165995762
QPE OrdC 350000 61250524998 349999 224953607
QPE OrdC 400000 80000599998 399999 295663167
QPE OrdC 450000 101250674998 449999 374494778
QPE OrdC 500000 125000749998 499999 462587339
QPE OrdD 50000 1250099996 49999 4595618
QPE OrdD 100000 5000199996 99999 18381496
QPE OrdD 150000 11250299996 149999 41450846
QPE OrdD 200000 20000399996 199999 73782311
QPE OrdD 250000 31250499996 249999 115293491
QPE OrdD 300000 45000599996 299999 165872498
QPE OrdD 350000 61250699996 349999 225915762
QPE OrdD 400000 80000799996 399999 294645292
QPE OrdD 450000 101250899996 449999 373213661
QPE OrdD 500000 125000999996 499999 460577333
QI1 Ale 50000 6790360 6439741 49624
QI1 Ale 100000 25703689 25006678 188608
QI1 Ale 150000 56166658 55086129 404134
QI1 Ale 200000 100927942 99505928 700352
```

```
QI1 Ale 250000 155578295 153798254 1118183
QI1 Ale 300000 230080189 227941489 1568334
QI1 Ale 350000 309699589 307204343 2267498
QI1 Ale 400000 402478987 399641416 2862923
QI1 Ale 450000 506619632 503392422 3457881
QI1 Ale 500000 624959666 621334928 4259262
QI1 OrdC 50000 10041164 9691284 69991
QI1 OrdC 100000 39613427 38913547 276459
QI1 OrdC 150000 88716937 87667057 623571
QI1 OrdC 200000 157351712 155951832 1100589
QI1 OrdC 250000 245517719 243767839 1719959
QI1 OrdC 300000 353214989 351115109 2480615
QI1 OrdC 350000 480443508 477993628 3367680
QI1 OrdC 400000 627203259 624403379 4397786
QI1 OrdC 450000 793494289 790344409 5573863
QI1 OrdC 500000 979316552 975816672 6874523
QI1 OrdD 50000 10041558 9716672 70600
QI1 OrdD 100000 39614212 38964326 277600
QI1 OrdD 150000 88718112 87743226 626420
QI1 OrdD 200000 157353278 156053392 1106055
QI1 OrdD 250000 245519676 243894790 1721351
QI1 OrdD 300000 353217336 351267450 2474494
QI1 OrdD 350000 480446246 478171360 3375226
QI1 OrdD 400000 627206388 624606502 4416920
QI1 OrdD 450000 793497808 790572922 5577304
QI1 OrdD 500000 979320462 976070576 6905243
QI5 Ale 50000 31010506 30776542 197350
QI5 Ale 100000 125496899 125028056 907842
QI5 Ale 150000 286790939 286118814 1959538
QI5 Ale 200000 509577104 508640261 3428241
QI5 Ale 250000 774058144 772810881 5217746
QI5 Ale 300000 1143222951 1141812029 8957351
QI5 Ale 350000 1531734961 1530104051 9377933
QI5 Ale 400000 1971345806 1969469075 16265725
QI5 Ale 450000 2605763863 2603641314 19968742
QI5 Ale 500000 3120692348 3118453013 19694267
QI5 OrdC 50000 39239102 38989128 274493
QI5 OrdC 100000 156603161 156103187 1097413
QI5 OrdC 150000 352092227 351342253 2478438
QI5 OrdC 200000 625706286 624706312 4411052
QI5 OrdC 250000 977445352 976195378 6900110
QI5 OrdC 300000 1407309411 1405809437 9938633
QI5 OrdC 350000 1915298477 1913548503 13531887
QI5 OrdC 400000 2501412536 2499412562 17674626
QI5 OrdC 450000 3165651602 3163401628 22386215
QI5 OrdC 500000 3908015661 3905515687 27637204
QI5 OrdD 50000 39240666 39015688 274848
QI5 OrdD 100000 156606288 156156310 1099645
```

```
QI5 OrdD 150000 352096916 351421938 2474125
QI5 OrdD 200000 625712538 624812560 4410640
QI5 OrdD 250000 977453166 976328188 6901909
QI5 OrdD 300000 1407318788 1405968810 9959305
QI5 OrdD 350000 1915309416 1913734438 13538131
QI5 OrdD 400000 2501425038 2499625060 17676446
QI5 OrdD 450000 3165665666 3163640688 22394612
QI5 OrdD 500000 3908031288 3905781310 27654127
QI10 Ale 50000 62245895 62061470 416946
QI10 Ale 100000 241288510 240893079 1700969
QI10 Ale 150000 595864137 595318298 3513211
QI10 Ale 200000 995494477 994779570 7919686
QI10 Ale 250000 1547743792 1546845158 13269068
QI10 Ale 300000 2100703004 2099589928 16723502
QI10 Ale 350000 3265346958 3264142422 21466563
QI10 Ale 400000 4059375497 4057920008 27942151
QI10 Ale 450000 5237617252 5235998072 39416629
QI10 Ale 500000 6241106692 6239280239 44242588
QI10 OrdC 50000 78253144 78053155 547663
QI10 OrdC 100000 312756269 312356280 2210634
QI10 OrdC 150000 703509394 702909405 4976437
QI10 OrdC 200000 1250512519 1249712530 8840306
QI10 OrdC 250000 1953765644 1952765655 13816561
QI10 OrdC 300000 2813268769 2812068780 19881596
QI10 OrdC 350000 3829021894 3827621905 27098376
QI10 OrdC 400000 5001025019 4999425030 35419434
QI10 OrdC 450000 6329278144 6327478155 44743357
QI10 OrdC 500000 7813781269 7811781280 55246088
QI10 OrdD 50000 78256270 78081278 549238
QI10 OrdD 100000 312762520 312412528 2202202
QI10 OrdD 150000 703518770 702993778 4965019
QI10 OrdD 200000 1250525020 1249825028 8839437
QI10 OrdD 250000 1953781270 1952906278 13813244
QI10 OrdD 300000 2813287520 2812237528 19883053
QI10 OrdD 350000 3829043770 3827818778 27068746
QI10 OrdD 400000 5001050020 4999650028 35359229
QI10 OrdD 450000 6329306270 6327731278 44806677
QI10 OrdD 500000 7813812520 7812062528 55301496
QNR Ale 50000 1057352 208351 9008
QNR Ale 100000 2279586 439085 19060
QNR Ale 150000 3539465 678448 29385
QNR Ale 200000 4822923 924738 40389
QNR Ale 250000 6067220 1174773 50822
QNR Ale 300000 7498916 1425953 61498
QNR Ale 350000 8745317 1684281 72333
QNR Ale 400000 10229921 1941584 84456
QNR Ale 450000 11464072 2205030 94741
QNR Ale 500000 12927268 2465525 106330
```

```
QNR OrdC 50000 750015 32767 3120
QNR OrdC 100000 1600016 65535 6561
QNR OrdC 150000 2456803 84464 12615
QNR OrdC 200000 3400017 131071 13646
QNR OrdC 250000 4250017 131071 16352
QNR OrdC 300000 5213588 168928 20348
QNR OrdC 350000 6213588 218928 24975
QNR OrdC 400000 7200018 262143 28800
QNR OrdC 450000 8100018 262143 31332
QNR OrdC 500000 9000018 262143 34232
QNR OrdD 50000 750028 57766 3314
QNR OrdD 100000 1600030 115534 6975
QNR OrdD 150000 2456820 159464 9852
QNR OrdD 200000 3400032 231070 13702
QNR OrdD 250000 4250032 256070 16409
QNR OrdD 300000 5213606 318928 20390
QNR OrdD 350000 6213606 393928 24790
QNR OrdD 400000 7200034 462142 28883
QNR OrdD 450000 8100034 487142 31391
QNR OrdD 500000 9000034 512142 34463
```