Αλγόριθμοι

Project

Αβραμίδης Πρόδρομος ΑΕΜ: 03291

Πίσχος Βασίλειος ΑΕΜ: 03175

Ερώτημα 1

Οι ψευδοκώδικες για τους αλγορίθμους:

- **Quicksort_1p_l**

```
quick_sort(array,start,end)
   if (start < end)
      pivot <- array[start];
      i <- start;
      j <- end;

      while(i<j)

         while(array[i] <= pivot and i < end)
            i <- i+1;
         end

         while(array[j] > pivot)
            j <- j-1;
         end

         if (i < j)
            swap(array[i], array[j]);
         end
      end
      quick_sort(array,start,j-1);
      quick_sort(array,j+1,end);
   end
end
```

- **Quicksort_1p_r**

```
quick_sort_random_one(array, start, end)
    if (start < end)
            random = (rand() modulo (end - start)) + end
            swap(array[start], array[random])

            i<-start
            j<-end
            pivot<-array[start]

            while(i<j)
                    while(array[i] <= pivot && i<end)
                            i <- i+1
                    end
                    while(array[j] > pivot)
                            j <- j+1
                    end

                    if(i<j)
                            swap(array[i], array[j])
                    end
            end

            swap(array[]start, array[j]);

            quick_sort_random_one(array, start, j-1)
            quick_sort_random_one(array, j+1, comparisons)
    end
end
```

- **Quicksort_1p_m**

```
quick_sort_random_median(array, start,end)

   if (start < end)
      random[0] <- (random_generator() modulo (end - start)) + start
      random[1] <- (random_generator() modulo (end - start)) + start
      random[2] <- (random_generator() modulo (end - start)) + start

      pivot_options[0] <- array[random[0]]
      pivot_options[1] <- array[random[1]]
      pivot_options[2] <- array[random[2]]

if  (pivot_options[0] >= pivot_options[1] and pivot_options[0] <=
                                                              pivot_options[2])
         pivot <- pivot_options[0]
         swap(array[start], array[random[0]])
      end

      if (pivot_options[1] >= pivot_options[0] and pivot_options[1] <=
                                                              pivot_options[2])
         pivot <- pivot_options[1]
         swap(array[start], array[random[1]])
      end

      if (pivot_options[2] >= pivot_options[1] and pivot_options[2] <=
                                                              pivot_options[0])
         pivot <- pivot_options[2]
         swap(array[start], array[random[2]])
      end

      i <- start
      j <- end
      pivot <- array[start]

      while(i<j)
```

```
        while(array[i] <=  pivot and i < end)
          i <- i+1
        end

        while(array[j] > pivot)
          j <- j-1
        end

        if (i < j)
          swap(array[i], array[j])
        end
      end
      swap(array[start], array[j])

      quick_sort_random_median(array,start,j-1)
      quick_sort_random_median(array,j+1,end)
    end
end
```

- **Quicksort_2p_r**

```
quick_sort_random_two(array, start, end)

    if (start < end)
            random[0] <- (rand() modulo (end-start)) + start;
            random[1] <- (rand() modulo (end-start)) + start;

            swap(array[start], array[random[0]])
            swap(array[end], array[random[1]])

            if (array[start > array[end])
                    swap(array[start], array[end])
            end

            pivot1 <- array[start]
            pivot2 <- array[end]

            l <- start + 1
            k <- l
            g <- end - 1

            while(k <= g)
                    if (array[k] < pivot1)
                            swap(array[k], array[l])
                            l <- l+1
                    end
                    else if (array[k] >= pivot)
                            while(array[g] > pivot2 && k < g)
                                    g <- g-1
                            end

                            swap(array[k], array[j])
                            g <- g-1

                            if (array[k] < pivot1)
```

```
                        swap(array[k], array[l])
                        l <- l+1
                    end
                end
                k <- k+1
            end

            l <- l-1
            g <- g+1

            swap(array[start], array[l])
            swap(array[end], array[g])

            quick_sort_random_two(array, start, l-1)
            quick_sort_random_two(array, l+1, g-1)
            quick_sort_random_two(start, g+1, end)
        end
    end
```

- **Quicksort_2p_r_pre**

```
Quick_sort_two_pre(array, size)
        pivot_size <- round(sqrt(size))
        potential_pivots <- generate_pivots(pivot, size)
        quick_sort(potential_pivots,0,pivot_size-1)

        quick_sort_two_pre_body(array, potential_pivots, counter_pivot,
pivot_size, 0, size-1)
        free(potential_pivots)
end

Quick_sort_two_pre_body(array, potential_pivots, counter_pivot, size, start, end)
        potential_pos <- counter_pivot

        if(start < end)
                random[0] = random_selection(potential_pivots, counter_pivot,
                                                        start, end, size, potential_pos)
                random[1] = random_selection(potential_pivots, counter_pivot,
                                                        start, end, size, potential_pos)

                if (random[0] = -1)
                        random[0] = (rand() modulo(end-start)) + start
                end
                if (random[1] = -1)
                        random[1] = (rand() modulo(end-start)) + start
                end

                swap(array[start], array[random[0])
                swap(array[end], array[random[1])

                if(array[start] > array[end])
                        swap(array[start], array[end])
                end
```

```
pivot1 = array[start]
pivot2 = array[end]

l <- start+1
k <- 1
g <- end-1

while(k <= g)
        if(array[k] < pivot1)
                swap(array[k], array[l])
                l <- l+1
        end
        else if (array[k] >= pivot2)
                while(array[g] > pivot2 && k < g)
                        g <- g-1
                end

                swap(array[k], array[g])
                g <- g-1

                if (array[k] <pivot1)
                        swap(array[k], array[l])
                        l <- l+1
                end
        end
        k <- k+1
end

l <- l-1
g <- g+1
swap(array[start], array[l])
swap(array[end], array[g])

Quick_sort_two_pre_body(array, potential_pivots, counter_pivot,
                                                        size, start, l-1)
```

```
            Quick_sort_two_pre_body(array, potential_pivots, counter_pivot,
                                                                    size, l+1, g-1)
            Quick_sort_two_pre_body(array, potential_pivots, counter_pivot,
                                                                  size, g+1, end)
        end
end
```

- **Quicksort_3p_r**

```
quick_sort_random_three(array,start,end)

   i = start + 2
   j = start + 2
   k = end - 1
   l = end - 1

   if (start < end)

      random[0] = (random_generator() % (end - start)) + start
      random[1] = (random_generator() % (end - start)) + start
      random[2] = (random_generator() % (end - start)) + start

      swap(array[start], array[random[0]])
      swap(array[start+1], array[random[1]])
      swap(array[end], array[random[2]])

      pivot1 <- array[start]
      pivot2 <- array[start+1]
      pivot3 <- array[end]

      if (pivot1 > pivot3)
         swap(array[start], array[end])
         swap(pivot1,pivot3)
      end

      if (pivot1 > pivot2)
         swap(array[start], array[start+1])
         swap(pivot1,pivot2)
      end

      if (pivot2 > pivot3)
         swap(array[end], array[start+1])
         swap(pivot2,pivot3)
```

```
end

while (j <= k)
   while (array[j] < pivot2 and j <<- k)
      if (array[j] < pivot1)
         swap(array[i], array[j])
         i <- i+1
      end
      j < j+1
   end
   while (array[k] > pivot2 and j <<- k)
      if (array[k] > pivot3)
         swap(array[k], array[l])
         l <- l-1
      end
      k <- k-1
   end
   if (j <= k)
      if (array[j] > pivot3)
         if (array[k] < pivot1)
            swap(array[j], array[i])
            swap(array[i], array[k])
            i <- i+1
         end
         else
            swap(array[j], array[k])
         end
         swap(array[k], array[l])
         j <- j+1
         k <- k-1
         l <- l-1
      end
      else
         if (array[k] < pivot1)
            swap(array[j], array[i])
```

```
            swap(array[i], array[k])
            i++
          end
          else
            swap(array[j], array[k])
          end
          j <- j+1
          k <- k-1
        end
      end
    end
    i <- i-1
    j <- j-1
    k <- k+1
    l <- l+1

    swap(array[start+1], array[i])
    swap(array[i], array[j])
    i <- i-1

    swap(array[start], array[i])
    swap(array[end], array[l])


    quick_sort_random_three(array,start,i-1)
    quick_sort_random_three(array,i+1, j-1)
    quick_sort_random_three(array,j+1, l-1)
    quick_sort_random_three(array, l+1, end)
end
```

- **Quicksort_3p_r_pre**

```
quick_sort_three_pre(array, size)
   counter_pivot  <- 0

   pivot_size <- round(sqrt(size))

   potential_pivots <-  generate_pivots(pivot_size, size)

   quick_sort(potential_pivots,0, pivot_size-1)

   quick_sort_three_pre_body(array, potential_pivots, counter_pivot,
                                                          pivot_size, 0, size-1)
end

random_selection(potential_pivots, counter_pivot, start ,end, size,
                                                          potential_pos)

   if (counter_pivot = size)
      return -1
   end

   while (potential_pos < size)
      random <- potential_pivots[potential_pos]
      if (random > end)
         return -1
      end
      if (random > start)
         potential_pivots[potential_pos] <- -1
         if (potential_pos = counter_pivot+1)
            counter_pivot <- counter_pivot+ 1
         end
         return random
      end
      potential_pos<-potential_pos + 1
   end
```

```
   return -1
end
quick_sort_three_pre_body(array, potential_pivots,counter_pivot, size,
                                                                    start, end)

   i <- start + 2
   j <- start + 2
   k <- end - 1
   l <- end - 1
   potential_pos <- counter_pivot

   if (start < end)
       random[0] <- random_selection(potential_pivots, counter_pivot, start,
                                                        end, size, potential_pos)
       random[1] <- random_selection(potential_pivots, counter_pivot, start,
                                                        end, size, potential_pos)
       random[2] <- random_selection(potential_pivots, counter_pivot, start,
                                                        end, size, potential_pos)

      if (random[0] = -1)
         random[0] <- (random_generator() % (end - start)) + start
      end
      if (random[1] = -1)

         random[1] <- (random_generator() % (end - start)) + start
      end
      if (random[2] = -1)

         random[2] <- (random_generator() % (end - start)) + start
      end

      swap(array[start], array[random[0]])
      swap(array[start+1], array[random[1]])
      swap(array[end], array[random[2]])
```

```
pivot1 <- array[start]
pivot2 <- array[start+1]
pivot3 <- array[end]

if (pivot1 > pivot3)
   swap(array[start], array[end])
   swap(pivot1,pivot3)
end

if (pivot1 > pivot2)
   swap(array[start], array[start+1])
   swap(pivot1,pivot2)
end

if (pivot2 > pivot3)
   swap(array[end], array[start+1])
   swap(pivot2,pivot3)
end

while (j <= k)
   while (array[j] < pivot2 and j <<- k)
      if (array[j] < pivot1)
         swap(array[i], array[j])
         i <- i+1
      end
      j < j+1
   end
   while (array[k] > pivot2 and j <<- k)
      if (array[k] > pivot3)
         swap(array[k], array[l])
         l <- l-1
      end
      k <- k-1
   end
```

```
   if (j <= k)
      if (array[j] > pivot3)
         if (array[k] < pivot1)
            swap(array[j], array[i])
            swap(array[i], array[k])
            i <- i+1
         end
         else
            swap(array[j], array[k])
         end
         swap(array[k], array[l])
         j <- j+1
         k <- k-1
         l <- l-1
      end
      else
         if (array[k] < pivot1)
            swap(array[j], array[i])
            swap(array[i], array[k])
            i <- i + 1
         end
         else
            swap(array[j], array[k])
         end
         j <- j+1
         k <- k-1
      end
   end
end
i <- i-1
j <- j-1
k <- k+1
l <- l+1

swap(array[start+1], array[i])
```

```
        swap(array[i], array[j])
        i <- i-1

        swap(array[start], array[i])
        swap(array[end], array[l])


            quick_sort_three_pre_body(array, potential_pivots, counter_pivot,
                                                                size, start, i-1)
            quick_sort_three_pre_body(array, potential_pivots, counter_pivot,
                                                                size, i+1, j-1)
            quick_sort_three_pre_body(array, potential_pivots, counter_pivot,
                                                                size, j+1, l-1)
            quick_sort_three_pre_body(array, potential_pivots, counter_pivot,
                                                                size, l+1, end)
    end
end
```

Ερώτημα 2

- **Quicksort_1p_l και Quicksort_1p_r**

Για το κανονικό quick sort πρέπει να ισχύει ότι σε κάθε αναδρομική κλήση τα στοιχεία αριστερά του pivot είναι μικρότερα από αυτό και δεξιά μεγαλύτερα.

- **Quicksort_1p_m**

Για το quick sort με median από τυχαία pivot πρέπει να ισχύει ότι σε κάθε αναδρομική κλήση τα στοιχεία αριστερά του pivot είναι μικρότερα από αυτό και δεξιά μεγαλύτερα και ότι το pivot είναι το median από τα τυχαία νούμερα.

- **Quicksort_2p_r και Quicksort_2p_r_pre**

Για το quick sort με δύο τυχαία pivot πρέπει να ισχύει σε κάθε αναδρομική κλήση ότι το αριστερά pivot είναι μικρότερο του δεξιά και ότι τα στοιχεία αριστερά του αριστερού pivot είναι μικρότερα του, ενώ δεξιά του μεγαλύτερά του αλλά μικρότερα του δεξιού pivot. Δεξιά του δεξιού pivot τα στοιχεία είναι μεγαλύτερα του.

- **Quicksort_3p_r και Quicksort_3p_r_pre**

Για το quick sort με τρία τυχαία pivot πρέπει να ισχύει σε κάθε αναδρομική κλήση ότι το αριστερά < μεσαίου < δεξιά pivot και ότι τα στοιχεία αριστερά του αριστερού pivot είναι μικρότερα του, ενώ δεξιά του μεγαλύτερά του, αλλά μικρότερα του μεσαίου pivot. Δεξιά του μεσαίου pivot τα στοιχεία είναι μεγαλύτερα του αλλά μικρότερα του δεξιά και τα στοιχεία δεξιά από το δεξιά pivot είναι μεγαλύτερα του.

Ερώτημα 3 και 4

Πρώτο μηχάνημα:

CPU: 6 cores 12, threads, base speed 2.7 GHz

Ram 16 GB, Speed 3200 MHz.

Average Time:

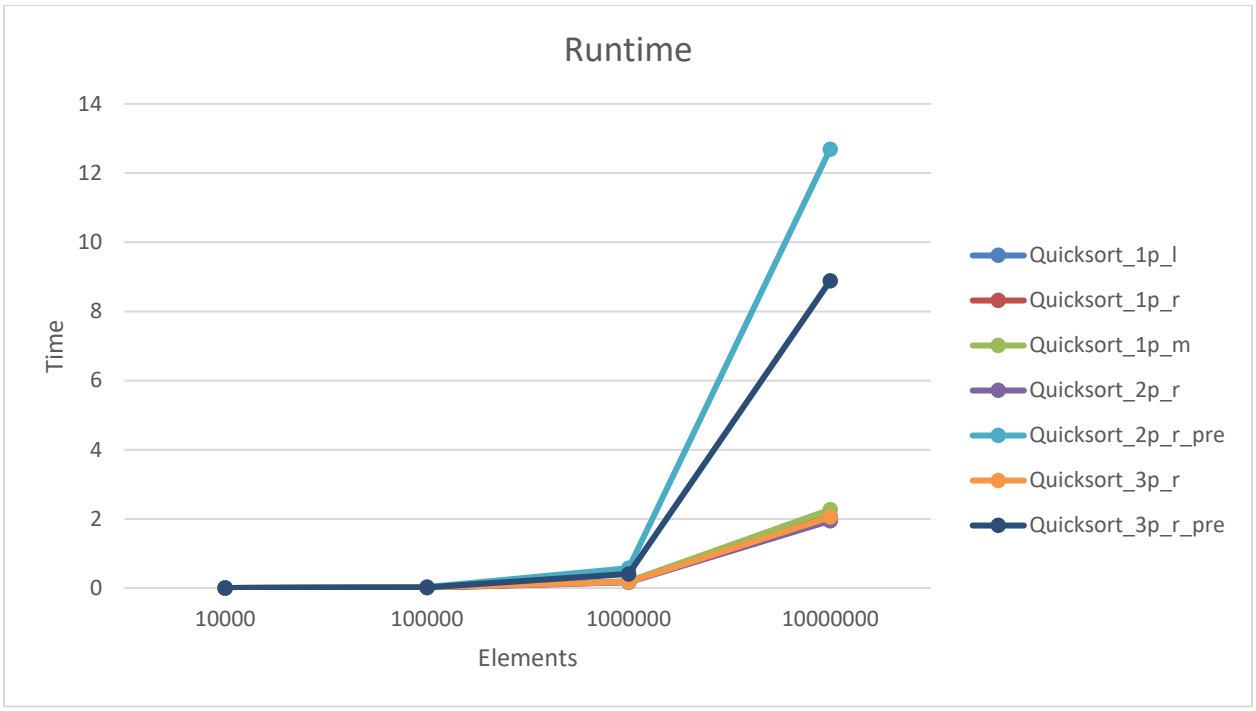
Elements	10000	100000	1000000	10000000
Quicksort_1p_l	0.00121972	0.015212	0.181746	2.025715
Quicksort_1p_r	0.00122	0.014994	0.17843	2.072736
Quicksort_1p_m	0.00141068	0.017059	0.197498	2.267759
Quicksort_2p_r	0.00113312	0.014164	0.167309	1.943553
Quicksort_2p_r_pre	0.00156916	0.027282	0.573396	12.69125
Quicksort_3p_r	0.00123	0.015017	0.177469	2.051903
Quicksort_3p_r_pre	0.0015666	0.022206	0.409229	8.884429

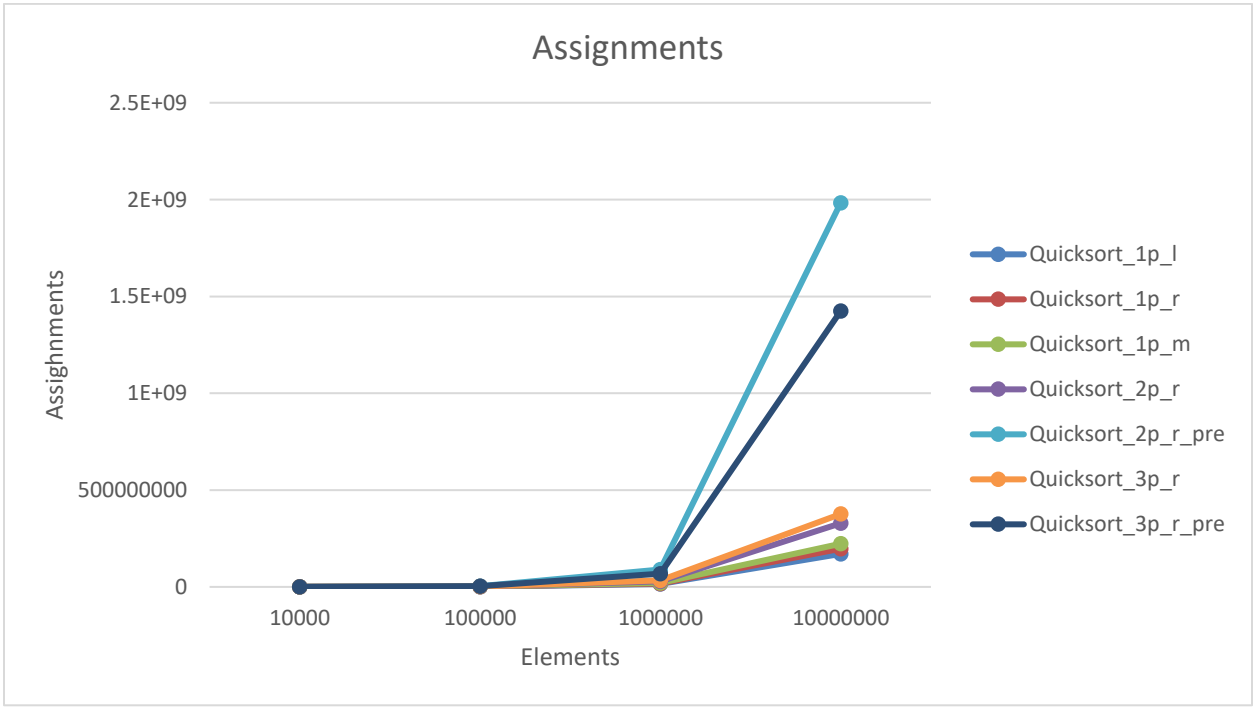
Average Assignments:

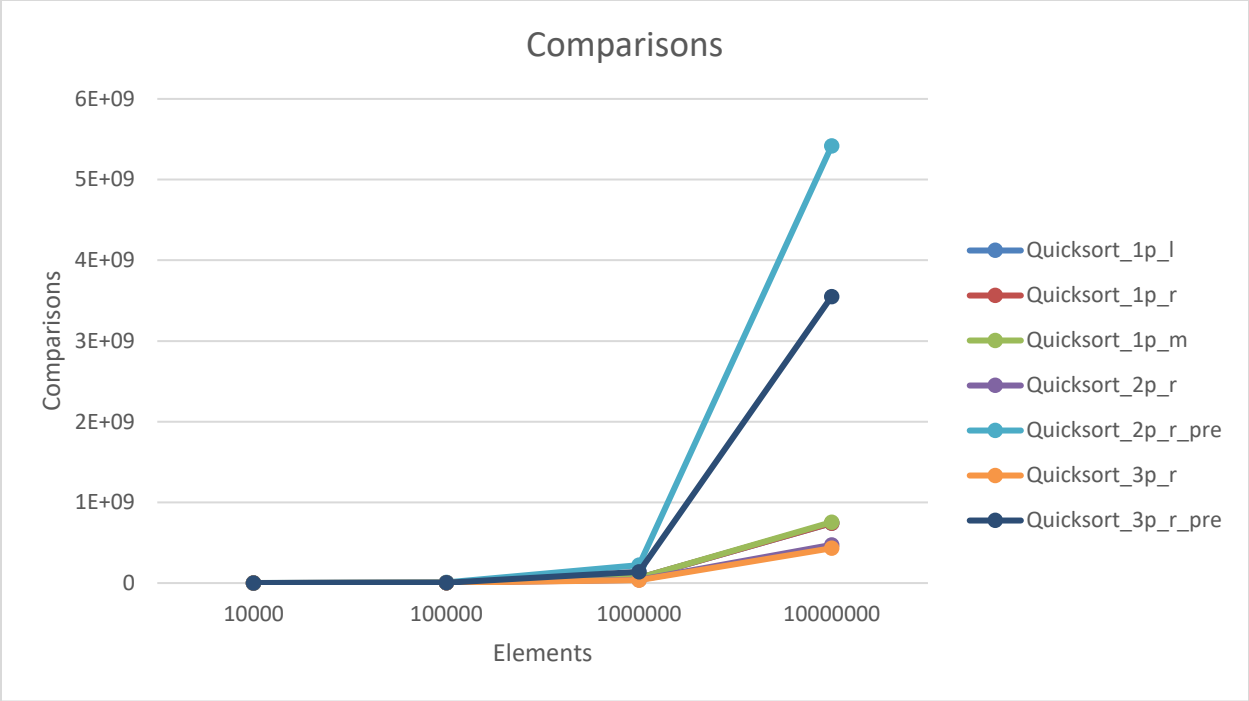
Elements	10000	100000	1000000	10000000
Quicksort_1p_l	101325.2	1245631	14729511	170460923.5
Quicksort_1p_r	127944	1511238	17418258	197255011.8
Quicksort_1p_m	152908.7	1760289	19948715	222878314.3
Quicksort_2p_r	204574.1	2466558	28910957	328665601.7
Quicksort_2p_r_pre	266232.1	4354088	89717553	1983712974
Quicksort_3p_r	241034	2853877	32923669	376434029.4
Quicksort_3p_r_pre	288380.8	3922619	68301483	1424828535

Average Comparisons:

Elements	10000	100000	1000000	10000000
Quicksort_1p_l	421509.4	5267565	63675659.22	742423308.7
Quicksort_1p_r	420837.2	5280600	63542054.86	742459210.6
Quicksort_1p_m	445991.1	5522716	65180848.38	754642599.5
Quicksort_2p_r	257560.5	3295452	40028415.14	473496551
Quicksort_2p_r_pre	429162.6	8844195	220866727.9	5416572998
Quicksort_3p_r	240747.3	3012117	36602104.24	431493903.6
Quicksort_3p_r_pre	347746.2	5883299	138966134.2	3550460422


Runtime
14
12
10
8
6
4
2
0
Time
10000
100000
1000000
10000000
Elements
Quicksort_1p_l
Quicksort_1p_r
Quicksort_1p_m
Quicksort_2p_r
Quicksort_2p_r_pre
Quicksort_3p_r
Quicksort_3p_r_pre


Assignments
2.5E+09
2E+09
1.5E+09
1E+09
500000000
0
Assighnments
10000
100000
1000000
10000000
Elements
Quicksort_1p_l
Quicksort_1p_r
Quicksort_1p_m
Quicksort_2p_r
Quicksort_2p_r_pre
Quicksort_3p_r
Quicksort_3p_r_pre


Comparisons
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
Comparisons
10000
100000
1000000
10000000
Elements
Quicksort_1p_l
Quicksort_1p_r
Quicksort_1p_m
Quicksort_2p_r
Quicksort_2p_r_pre
Quicksort_3p_r
Quicksort_3p_r_pre

Δεύτερο μηχάνημα:

CPU: 6 cores 6, threads, base speed 3.7 GHz

Ram 16 GB, Speed 3200 MHz.

Time Variance:

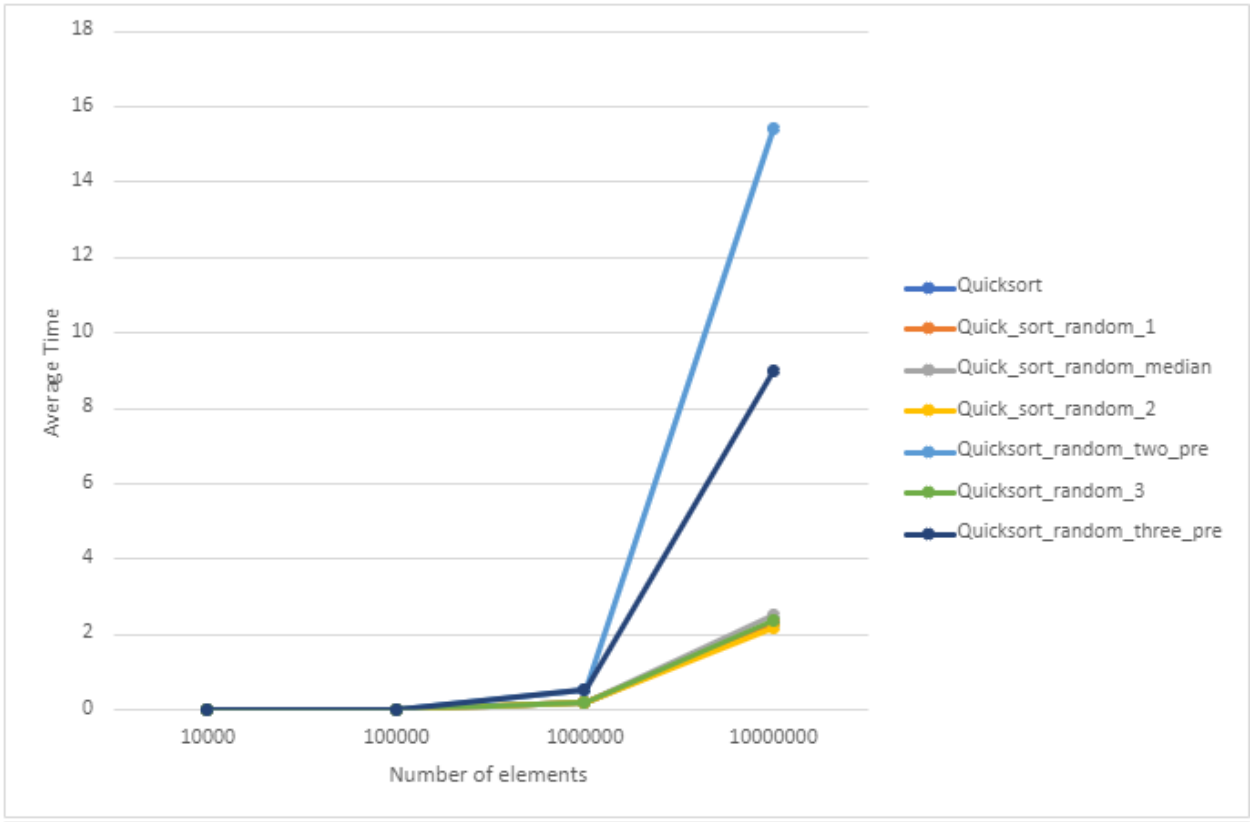
Elements	10000	100000	1000000	10000000
Quicksort_1p_l	3.48E-08	3.48396E-08	9.26198E-06	8.3604E+13
Quicksort_1p_r	2.70846E-09	4.43945E-08	2.00049E-06	7.87888E+13
Quicksort_1p_m	2.83458E-09	7.03641E-08	7.55131E-06	7.52897E+13
Quicksort_2p_r	1.11926E-08	9.65562E-08	4.24132E-06	8.76396E+13
Quicksort_2p_r_pre	2.49213E-07	0.000278617	0.28623486	3.00016E+19
Quicksort_3p_r	1.27071E-08	5.82964E-08	4.18762E-06	1.83712E+13
Quicksort_3p_r_pre	1.22951E-07	0.000187546	0.164966215	1.59952E+19

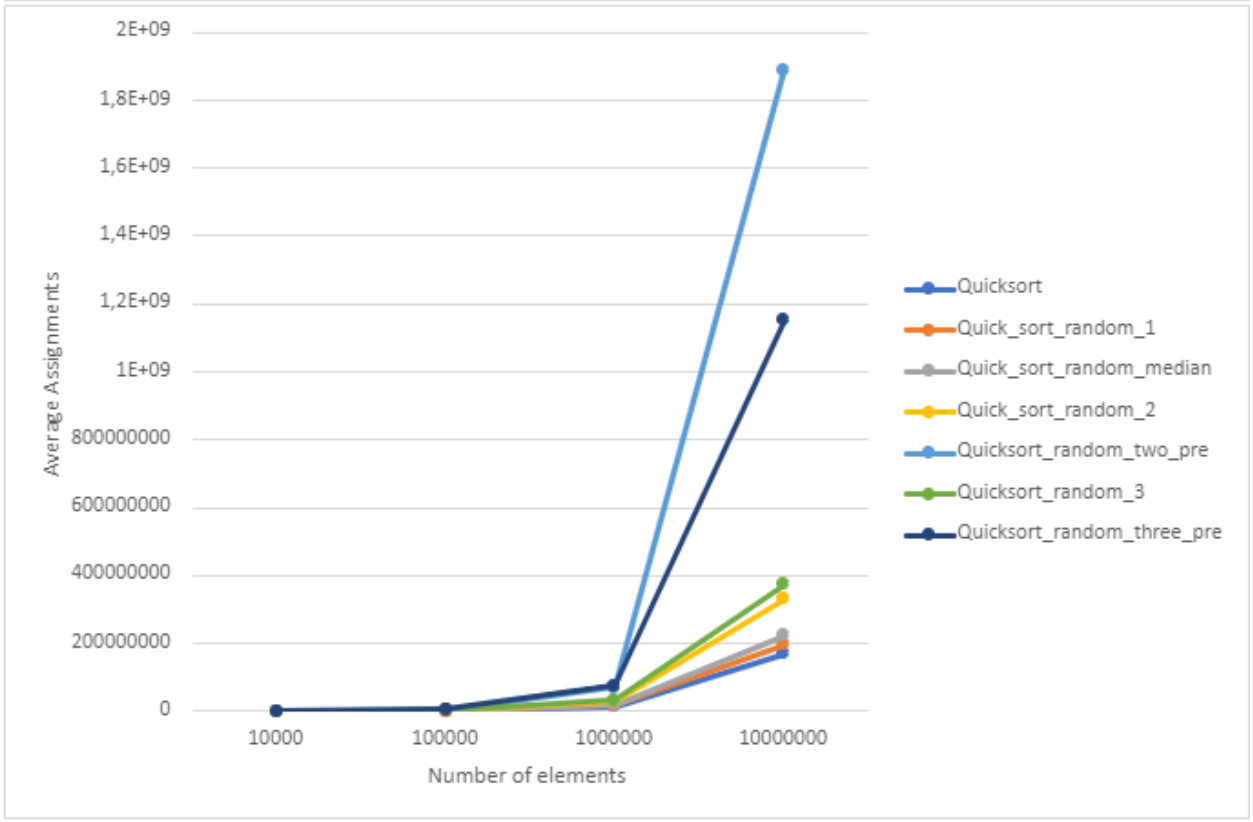
Assignments Variance:

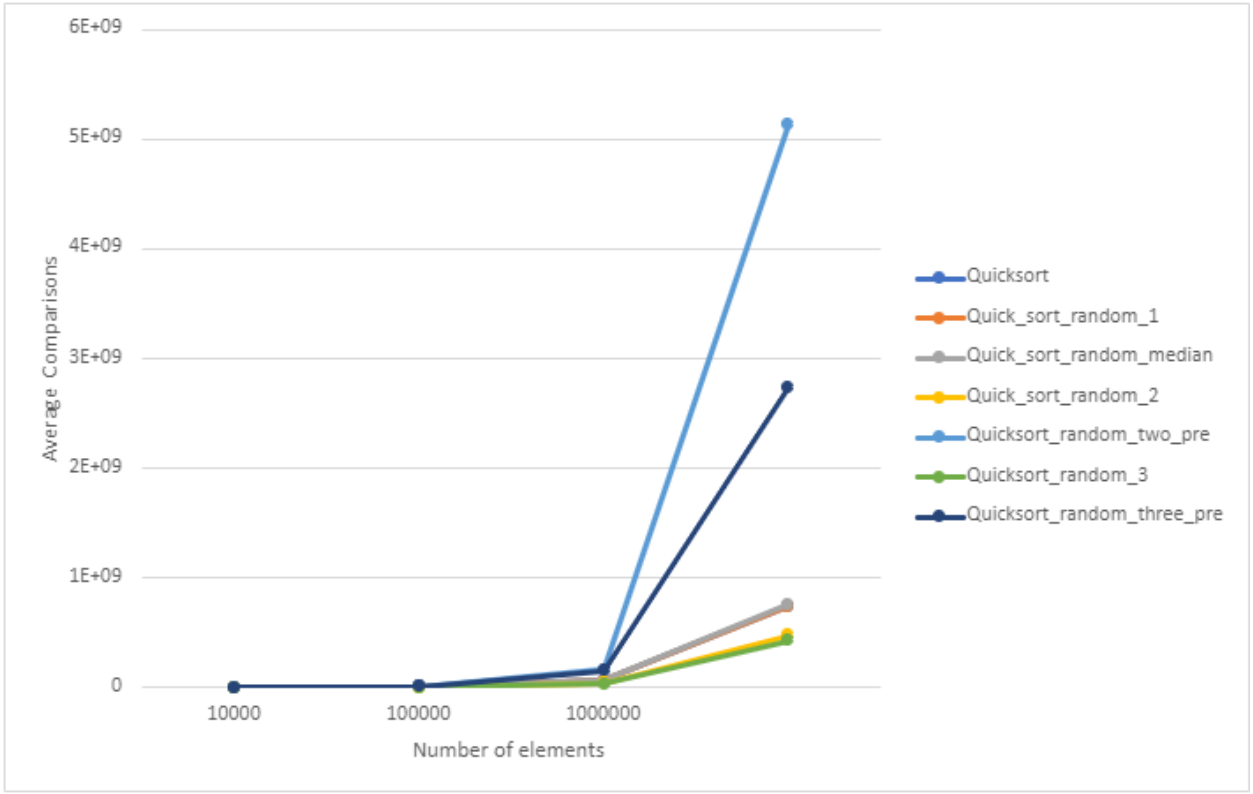
Elements	10000	100000	1000000	10000000
Quicksort_1p_l	664855.6506	46276501	4226168177	4.4857E+11
Quicksort_1p_r	648551.669	55769214	4793173963	3.9522E+11
Quicksort_1p_m	362196.6678	45020205.7	4710476255	3.4274E+11
Quicksort_2p_r	119312799.2	6710249572	7.923E+11	9.7725E+13
Quicksort_2p_r_pre	3532905919	3.8714E+12	3.8617E+15	3.332E+18
Quicksort_3p_r	98043510.95	7841704218	7.3575E+11	6.7967E+13
Quicksort_3p_r_pre	1546983037	2.733E+12	2.3375E+15	1.7803E+18

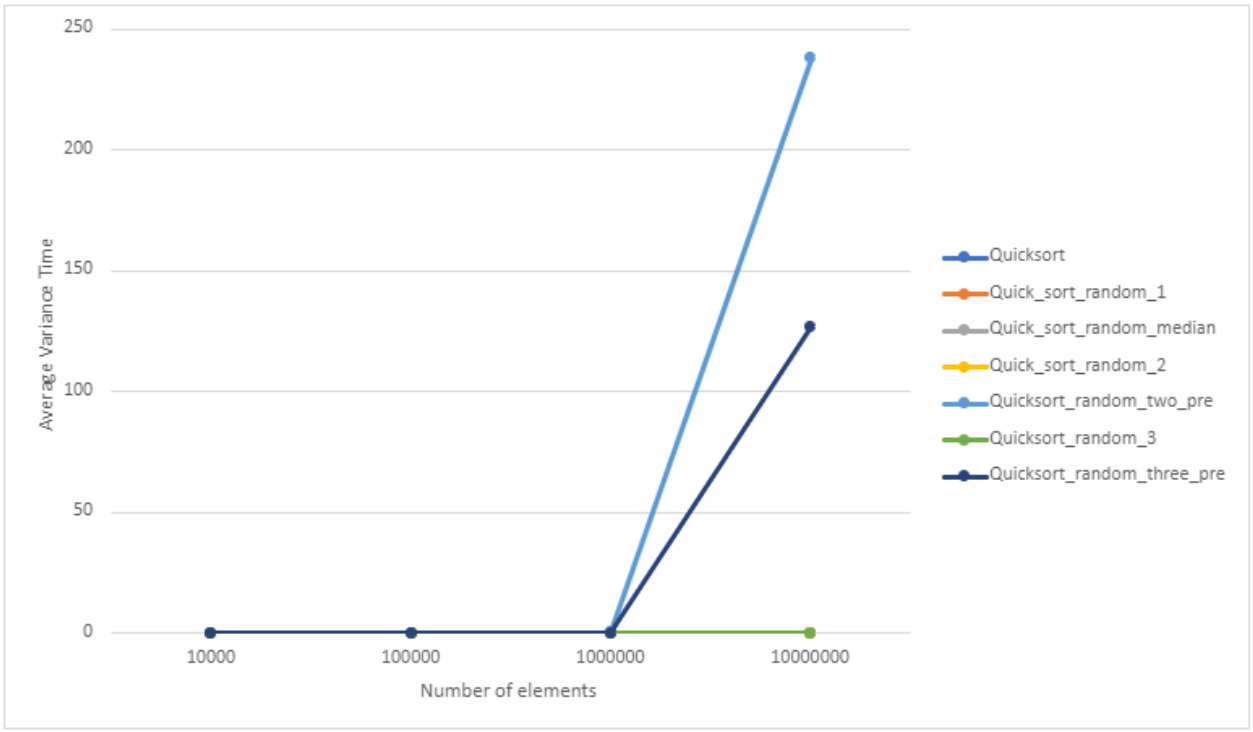
Comparisons Variance:

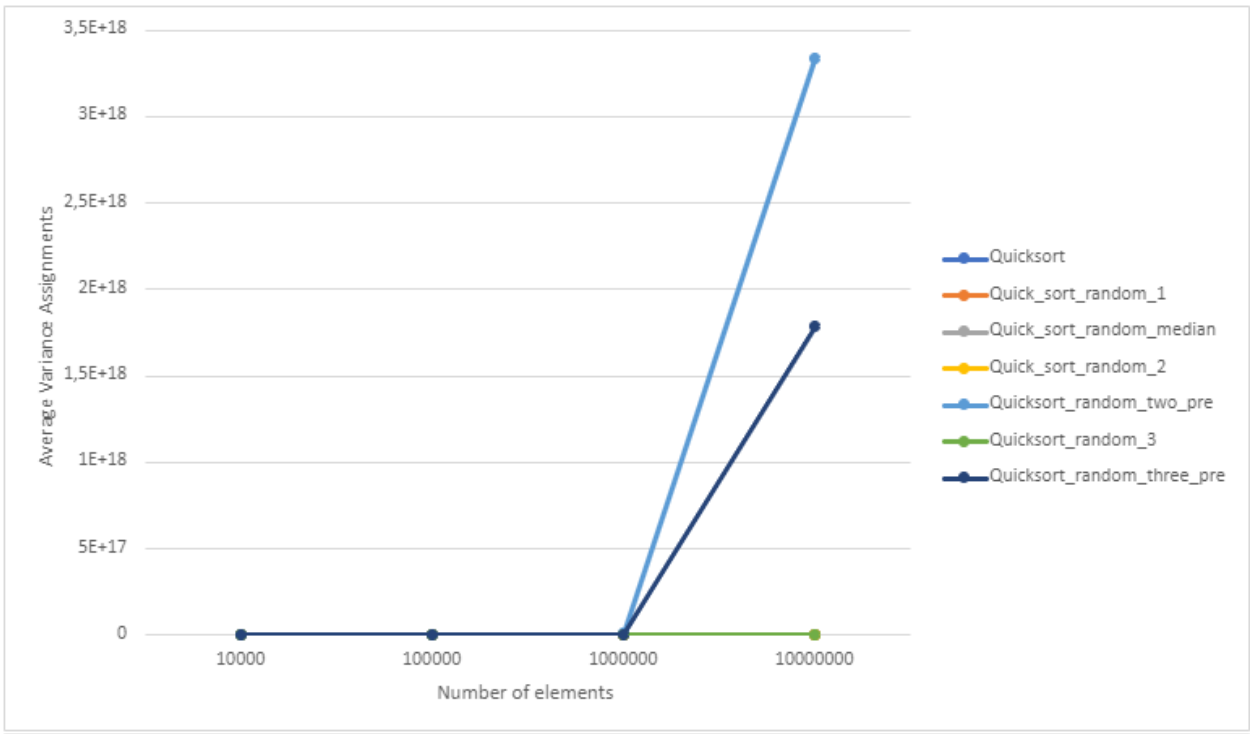
Elements	10000	100000	1000000	10000000
Quicksort_1p_l	119961911.8	6729595710	8.25842E+11	8.3604E+13
Quicksort_1p_r	109305720.2	11651163056	8.42166E+11	7.87888E+13
Quicksort_1p_m	62801250.2	10545578535	5.55607E+11	7.52897E+13
Quicksort_2p_r	83466817.96	5183177400	4.80288E+11	8.76396E+13
Quicksort_2p_r_pre	32030482984	3.47156E+13	3.47399E+16	3.00016E+19
Quicksort_3p_r	39099325.21	5315789645	4.99115E+11	1.83712E+13
Quicksort_3p_r_pre	12677288672	2.43351E+13	2.08726E+16	1.59952E+19

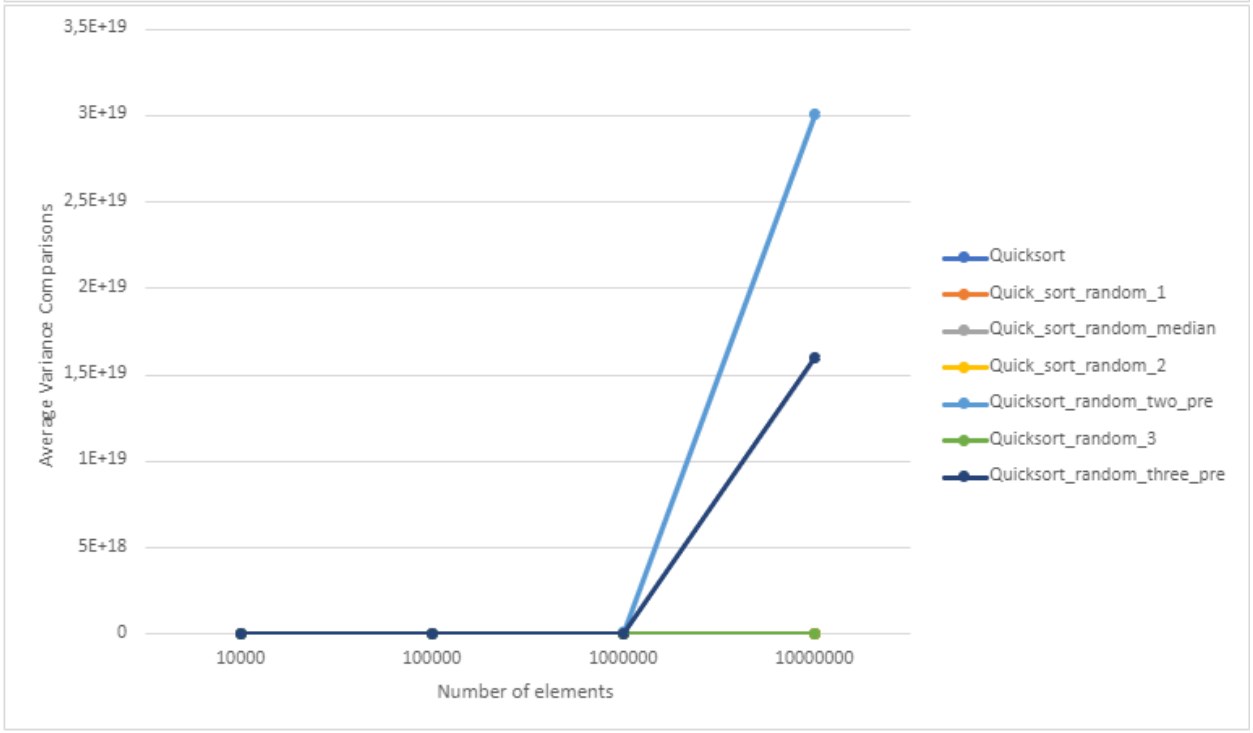

18
16
14
12
10
8
6
4
2
0
Average Time
10000
100000
1000000
10000000
Number of elements
Quicksort
Quick_sort_random_1
Quick_sort_random_median
Quick_sort_random_2
Quicksort_random_two_pre
Quicksort_random_3
Quicksort_random_three_pre


2E+09
1,8E+09
1,6E+09
1,4E+09
1,2E+09
1E+09
800000000
600000000
400000000
200000000
0
Average Assignments
10000
100000
1000000
10000000
Number of elements
Quicksort
Quick_sort_random_1
Quick_sort_random_median
Quick_sort_random_2
Quicksort_random_two_pre
Quicksort_random_3
Quicksort_random_three_pre


6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
Average Comparisons
10000
100000
1000000
Number of elements
Quicksort
Quick_sort_random_1
Quick_sort_random_median
Quick_sort_random_2
Quicksort_random_two_pre
Quicksort_random_3
Quicksort_random_three_pre


250
200
150
100
50
0
Average Variance Time
10000
100000
1000000
10000000
Number of elements
Quicksort
Quick_sort_random_1
Quick_sort_random_median
Quick_sort_random_2
Quicksort_random_two_pre
Quicksort_random_3
Quicksort_random_three_pre





Από τα γραφήματα βλέπουμε ότι οι αλγόριθμοι τρέχουν παρόμοια και στα δύο μηχανήματα.

Ερώτημα 5

- **Quicksort_1p_l**

Το quick sort με το αριστερό στοιχείο ως pivot είναι ένας αποδοτικός divide and conquer αλγόριθμος. Διαιρεί τον πίνακα σε δύο μέρη με το αριστερά μικρότερο του Pivot και το δεξιά μεγαλύτερο. Είναι αποδοτικός και για μεγάλα μεγέθη εισόδου.

- **Quicksort_1p_r**

Το quick sort με τυχαίο στοιχείο ως pivot έχει την ίδια λογική με το απλό, και είναι πολύ ελάχιστα πιο αργό από το κανονικό εξαιτίας των επιπλέων πράξεων για να επιλεχθεί τυχαία όμως για την χειρότερη περίπτωση του κανονικού που είναι να είναι να έχουμε ήδη sorted πίνακα δεν χαλάει η απόδοση του. Το πλεονέκτημα αυτό το κάνει καλύτερη επιλογή από το απλό παρά την πάρα πολύ μικρή πτώση στην απόδοση. Είναι αποδοτικός και για μεγάλα μεγέθη εισόδου.

- **Quicksort_1p_m**

Το quick sort με το median από τρία τυχαία στοιχεία ως pivot την ίδια λογική με το ένα τυχαίο χωρίς median όμως αν και το pivot είναι καλύτερης ποιότητας γιατί διαμερίζει καλύτερα τον πίνακα είναι λίγο ποιο αργό από αυτό λόγω της επιπλέον πολυπλοκότητας. Έχει το ίδιο πλεονέκτημα για τον ήδη sorted πίνακα. Είναι αποδοτικός και για μεγάλα μεγέθη εισόδου.

- **Quicksort_2p_r**

Το quick sort με δύο τυχαία στοιχεία ως pivot χωρίζει τον πίνακα κάθε αναδρομική κλήση σε 3 μέρη με τα στοιχεία μικρότερα από το αριστερό pivot αριστερά τα ενδιάμεσα στοιχεία ενδιάμεσα και τα μεγαλύτερα δεξιά από το δεξιά pivot. Έτσι ο πίνακας διαμερίζετε αποτελεσματικά σε μέρη που απαιτούν παρόμοιο χρόνο για να γίνουν sort και έτσι είναι ο ταχύτερος αλγόριθμος. Είναι αποδοτικός και για μεγάλα μεγέθη εισόδου.

- **Quicksort_3p_r**

Το quick sort με τρία τυχαία στοιχεία ως pivot χωρίζει τον πίνακα κάθε αναδρομική κλήση σε 4 μέρη αυτό επιτυγχάνει πολύ καλό χωρισμό του

πίνακα σε μέρη που απαιτούν παρόμοιο χρόνο για να γίνουν sort. Όμως είναι λίγο πιο αργό από την παραλλαγή με τα δύο pivot. Συμπεραίνουμε ότι το να βάζουμε παραπάνω pivot δεν αυξάνει πάντα την ταχύτητα του αλγορίθμου μας άρα δεν αξίζει η επιπλέον πολυπλοκότητα.

- **Quicksort_2p_r_pre και Quicksort_3p_r_pre**

Οι δύο παραλλαγές είναι ίδιες με τα κανονικά αλλά παίρνουν τα pivots από ένα πίνακα. Η επιλογή των pivot από τον πίνακα είναι πολύπλοκη και απαιτεί πολλές συγκρίσεις συνεπώς τρέχουν και οι δύο πιο αργά από τα κανονικά με τα 3 pivot να τρέχουν λίγο πιο γρήγορα από τα 2. Έχουν και οι δύο πολύ μεγάλο variance και δεν έχουν πλεονεκτήματα σε σχέση με τα κανονικά. Αρά είναι μη αποδοτικές. Η αποδοτικότητα μειώνετε ακόμα πιο πολύ για εισόδους με πολλά στοιχεία.

Πηγές

Yaroslavskiy's Dual-Pivoting Algorithm, για την παραλαγη των δυο pivot.

https://cs.uwaterloo.ca/~skushagr/multipivotQuicksort.pdf, για την παραλαγη των τριων pivot.

https://beginnersbook.com/2015/02/quicksort-program-in-c/, για το απλο Quicksort.