

CS263

ASSIGNMENT 4

**NAME:**

ARCHIT AGRAWAL

**ROLL NO.:**

202052307

**SECTION:**

A

## **Problem 1**

Consider two scenarios:

**Case(a):** Consider sorting a list of 1000 persons by the month they were born. There are only 12 possible months; if they are evenly spread throughout our array, we can anticipate each month to appear several times. As a result, we have an array with a lot of duplicate values. Situations like these occur frequently in real-life circumstances.

Think about how partitioning works in the quicksort for a moment – in particular, think about how elements equal to the pivot move during this process.

For example, 1, 4, 2, 4, 2, 4, 1, 2, 4, 1, 2, 2, 6, 9, 8, 9, 2, 2, 4, 1, 4, 4, 4.

If 4 is picked as a pivot in Simple Quick Sort, we fix only one 4 and recursively process remaining occurrences.

After one partition, we will have 1, 2, 2, 1, 2, 1, 2, 2, 2, 2, 1, 4, 4, 4, 4, 4, 4, 4, 4, 6, 9, 8, 9

Now think about the potential offered by modifying our partitioning technique to group array elements into three different groups: that is called 3-way partition.

**Case(b):** Consider another scenario, where we can have unique elements, can we apply a 3-way partition. If yes what will be our partition technique?

For example: 24, 8, 42, 75, 29, 77, 38, 57

Take two pivot elements and partition in 3 sub-lists. Suppose pivots are 24 and 57. Then we have three partition:- 8, 24, 42, 29, 38, 57, 75, 77

### **Algorithm (Case (a))**

The three-way quicksort is similar, but there are three sections. array arr[1 to n] is divided into three parts.

1. arr[1 to i]
2. arr[i + 1, j]
3. arr[j + 1, n]

Partition in this case can be done in the below mentioned way:

- Choose a pivot (say the last element everytime)
- Now, partition the array into three subarrays, one that contains all elements smaller than the pivot element (array index 1 to i), second which contains all the elements equal to the pivot element (array index i + 1 to j) and third which contains all the elements that are greater than the pivot element (array index j + 1 to n).
- Now, we have three subarrays, out of which one is sorted and placed correctly. Now, we can recur down in the remaining two subarrays (one that has elements smaller than pivot and other that has elements greater than the pivot).

### **Code (Case (a))**

```
import java.util.*;
public class QuickSort3Way {

    static int i, j;

    //method to swap two elements of array
    public static void swap(int[] arr, int i, int j){
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }

    //method to create partition for quicksort

    public static void partition(int[] arr, int l, int r){
        i = l - 1; j = r;
```

```
        if (r - l <= 1) {
            if (arr[r] < arr[l])
                swap(arr, r, l);
            i = l;
            j = r;
            return;
        }
        int mid = l;
        int pivot = arr[r];
        while (mid <= r) {
            if (arr[mid] < pivot)
                swap(arr, l++, mid++);
            else if (arr[mid] == pivot)
                mid++;
            else if (arr[mid] > pivot)
                swap(arr, mid, r--);
        }
        i = l-1;
        j = mid;
    }

    //method to quicksort
    public static void quicksort(int[] arr, int left, int right) {
        if (left >= right)
            return;

        partition(arr, left, right);
        //recursive calls
        quicksort(arr, left, i);
        quicksort(arr, j, right);
    }

    public static void display(int[] arr){
        for(int x = 1; x <= arr.length; x++){

            System.out.print(arr[x-1] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args){
        Scanner Sc = new Scanner(System.in);
        int[] a = new int[1000];
        Random Ob = new Random();
        for(int i = 0; i < 1000; i++){
            a[i] = Ob.nextInt(12) + 1;
        }
    }
}
```



or  $T(n) = 2T(n/2) + n \log n$  (neglecting 1 and 2)

$$\log_b a = \log_2 2 = 1 \text{ and } k = 1, p = 0$$

Solving using Master's Method gives

$$T(n) = \theta(n \log n)$$

The worst case still has a time complexity of  $O(n^2)$ . But in the average the case the complexity reduces to  $O(n \log n)$ . This algorithm serves good when there are a large number of repeated elements. In such cases the complexity approaches  $O(n)$ , however, it is  $O(n)$  only in the best case i.e the array has only 1 unique element.

### **Algorithm (Case (b))**

The idea of dual pivot quick sort is to take two pivots, one in the left end of the array and the second, in the right end of the array. The left pivot must be less than or equal to the right pivot, so we swap them if necessary.

Then, we begin partitioning the array into three parts: in the first part, all elements will be less than the left pivot, in the second part all elements will be greater or equal to the left pivot and also will be less than or equal to the right pivot, and in the third part all elements will be greater than the right pivot. Then, we shift the two pivots to their appropriate positions as we see in the below bar, and after that we begin quicksorting these three parts recursively, using this method.

### **Code (Case (b))**

```
import java.util.*;
public class DualPivotQuickSort {

    //method to swap two elements of array
    public static void swap(int[] arr, int i, int j){
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

```
        arr[j] = temp;
    }

    static void dualPivotQuickSort(int[] arr, int low, int high){

        if (low < high){

            // piv[] stores left pivot and right pivot.
            // piv[0] means left pivot and
            // piv[1] means right pivot
            int[] piv;
            piv = partition(arr, low, high);

            dualPivotQuickSort(arr, low, piv[0] - 1);
            dualPivotQuickSort(arr, piv[0] + 1, piv[1] - 1);
            dualPivotQuickSort(arr, piv[1] + 1, high);
        }
    }

    //method to create partition for quicksort

    public static int[] partition(int[] arr, int low, int high){

        //if lower pivot is at higher index, then swap the pivots
        if (arr[low] > arr[high])
            swap(arr, low, high);

        // p is the left pivot, and q
        // is the right pivot.
        int j = low + 1;
        int g = high - 1, k = low + 1;
        int p = arr[low], q = arr[high];

        while (k <= g) {

            // If elements are less than the left pivot
            if (arr[k] < p){
                swap(arr, k, j);
                j++;
            }

            // If elements are greater than or equal
            // to the right pivot
            else if (arr[k] >= q){

                while (arr[g] > q && k < g) g--;
            }
        }
    }
}
```

```
        swap(arr, k, g);
        g--;

        if (arr[k] < p){
            swap(arr, k, j);
            j++;
        }

        k++;
    }
    j--;
    g++;

    // Bring pivots to their appropriate positions.
    swap(arr, low, j);
    swap(arr, high, g);

    // Returning the indices of the pivots
    // because we cannot return two elements
    // from a function, we do that using an array.
    return new int[] {j, g};
}

public static void display(int[] arr){
    for(int i = 1; i <= arr.length; i++){
        if(i % 50 == 0) System.out.println();
        System.out.print(arr[i-1] + " ");
    }
    System.out.println();
}

public static void main(String[] args){
    Scanner Sc = new Scanner(System.in);
    int[] a = new int[500];

    Random rand = new Random();

    for(int i = 0; i < 500; i++){
        a[i] = rand.nextInt(1000) + 1;
    }

    dualPivotQuickSort(a, 0, a.length-1);
    System.out.println("SORTED ARRAY");
    System.out.println();
    display(a);
}
```



}

### Output (Case(b))

```

input
SORTED ARRAY
5 7 8 11 12 16 20 23 28 29 33 36 36 39 39 41 42 43 43 45 48 50 51 51 53 56 62 67 67 67 68 69 72 73 74 77 77
82 83 84 86 88 91 94 97 99 101 105 106
107 109 110 110 112 112 112 113 114 121 121 122 122 123 124 125 128 131 136 137 148 148 149 149 153 156 156
158 165 169 170 174 174 175 177 178 181 182 182 183 187 188 188 194 195 196 200 202 205 211
214 215 215 215 215 218 219 219 220 222 223 229 230 231 233 236 242 242 243 243 246 246 247 250 251 255 256
256 257 260 264 266 266 266 269 271 272 273 275 275 275 278 281 281 284 289 298 301 304 307
308 314 316 317 318 318 321 322 320 329 330 332 336 337 337 343 343 347 348 348 349 351 355 355 355 356 356 357
358 360 361 362 364 366 366 368 369 371 371 372 374 374 378 379 384 384 385 386 387 388 389
390 392 392 394 401 406 409 410 410 411 411 412 412 414 417 418 420 421 422 422 422 424 426 426 428 435
441 446 453 454 455 455 457 458 463 463 465 470 472 474 476 479 480 483 483 485 485 486
488 490 492 492 493 494 494 495 499 499 499 504 506 507 508 509 510 511 512 512 519 519 527 532 532 536 539
539 542 550 551 551 553 555 556 557 558 561 561 563 563 564 564 566 569 572 573 576 579 581
582 584 584 593 593 595 598 599 602 603 603 606 606 608 612 615 615 617 620 622 622 624 629 627 627 631 633
633 634 634 635 638 638 639 639 639 643 646 647 650 651 652 653 655 657 658 660 661 666 666
666 669 669 669 673 676 677 678 678 680 682 684 684 685 685 686 687 687 687 688 688 694 694 697 699 700 700 704
705 706 707 712 712 715 715 716 720 726 728 728 733 736 736 737 739 739 740 742 745 748 748
750 752 753 758 759 763 764 768 773 774 780 782 784 787 788 789 790 791 795 795 797 797 799 801 802 804 804
804 809 809 811 811 813 816 817 827 827 830 834 839 839 843 844 849 851 851 852 862 867 869
873 873 876 876 877 884 885 886 887 889 889 893 893 894 896 904 905 906 907 908 909 914 916 919 920 926 929
931 932 936 939 941 950 951 956 957 960 960 967 967 968 972 973 974 982 983 983 986 988 989
995
...Program finished with exit code 0
Press ENTER to exit console.

```

### Analysis (Case (b))

The recurrence relation for dualPivotQuickSort is:

$$T(n) = 3T\left(\frac{n}{3}\right) + n^2$$

Solving using Master's Theorem:

$$\log_3 3 = 1 \text{ and } k = 2 \text{ and } p = 0$$

$$\text{Therefore, } T(n) = O(n^2)$$

In worst case the time complexity is  $O(n^2)$  but there is a very rare chance that worst case takes place. In average and best case, the time complexity is  $O(n \log n)$ .

## Problem 2

Implement Randomized Quicksort, and analyse its best, worst, and average-case complexity. Show the graph with varying its input size

## Code

```
import java.util.*;

public class Main{

    public static void swap(int arr[], int i, int j){
        int temp = arr[j];
        arr[j] = arr[i];
        arr[i] = temp;
    }

    public static void random(int arr[], int low, int high){

        Random rand = new Random();
        int pivot = rand.nextInt(high - low) + low;

        swap(arr, pivot, high);
    }

    public static int partitionRandom(int arr[], int l, int h){

        random(arr, l, h);
        int pivot = arr[l];

        int i = h + 1;

        for(int j = h; j > l; j--){
            if(arr[j] > pivot){
                i--;
                swap(arr, i, j);
            }
        }

        swap(arr, i - 1, l);
        return i - 1;
    }

    public static void randomisedQuickSort(int arr[], int low, int high){

        if(low < high){
            int x = partitionRandom(arr, low, high);
            randomisedQuickSort(arr, low, x - 1);
        }
    }
}
```

```
        randomisedQuickSort(arr, x + 1, high);
    }

}

public static void main(String[] args){

    int[] arr = new int[500];
    Random rand = new Random();

    for(int i = 0; i < 500; i++){
        arr[i] = rand.nextInt(1000);
    }

    randomisedQuickSort(arr, 0, 499);


    for(int i = 0; i < 500; i++){

        System.out.print(arr[i] + " ");

    }

}
```

## Output



```
2 3 4 5 7 7 10 14 14 14 16 19 22 23 29 30 32 34 34 37 37 44 45 48 51 54 57 57 58 63 65 68 69 69 69 71 73 76
77 78 80 92 94 99 99 100 102 103 108 109 109 109 111 112 114 117 117 118 118 119 120 123 129 130 133 135 136
137 137 139 142 144 151 155 155 156 156 157 159 160 163 167 169 170 172 172 180 181 182 183 183 187 188 191
192 192 192 194 196 197 199 201 201 206 207 208 208 213 216 217 222 223 228 229 229 231 234 239 240 240 240
240 241 243 244 245 247 251 253 253 256 258 260 261 265 265 266 267 267 270 272 287 289 292 293 298 298 299
300 301 301 302 303 304 308 311 313 313 316 321 323 326 328 330 330 331 333 334 337 337 338 340 341 341 343
344 345 347 348 349 351 354 357 358 359 359 360 362 363 368 372 383 390 390 392 395 397 405 409 410 411 412
413 415 415 417 418 419 422 422 424 425 429 430 430 431 433 438 442 443 448 449 454 462 464 468 469 470 482
484 487 490 491 492 495 496 498 502 503 504 509 510 510 513 518 522 522 524 529 532 532 545 548 550 551 554
556 559 561 563 563 563 564 565 566 567 569 571 573 576 578 578 583 586 589 597 597 601 602 606 607 608 613
613 613 613 617 619 620 620 623 623 624 625 625 626 629 632 632 632 635 636 636 638 638 647 649 650 654 655
655 656 656 657 661 662 662 663 664 667 667 668 668 669 676 680 681 683 686 689 690 694 695 703 706 716 716
717 718 718 720 720 721 721 722 722 722 723 724 725 725 725 727 731 731 732 733 734 738 738 740 740 743 744
745 746 747 747 749 753 762 763 763 765 765 769 769 781 785 790 790 790 790 794 790 798 804 805 809 811 812
815 815 818 820 820 822 825 827 828 829 833 833 834 837 841 850 850 851 852 853 854 855 855 855 856 857 857
858 859 859 860 862 865 866 867 867 868 870 871 874 874 875 875 882 884 884 890 891 893 895 897 898 899 900
900 902 902 903 904 906 907 907 907 914 918 920 922 928 928 933 935 937 938 938 940 944 946 947 948 950 950
951 958 960 963 963 963 965 968 969 972 974 974 979 979 981 985 985 986 992 992 992 994 994 994 995 996 997
997

...Program finished with exit code 0
Press ENTER to exit console.
```

## Analysis

Number of elements in array	Time taken in milliseconds(ms)
100	0.102
1000	0.277
10000	3.882
100000	22.451

<b>Best Case</b>	<b>Average Case</b>	<b>Worst Case</b>
$O(n \log n)$	$O(n \log n)$	$O(n^2)$

