1. the indexes i and j never go beyond the array boundaries. Why?

R:

Pentru prima iteratie se va interschimba mereu primul element si un element mai mic sau egal decat el (in cel mai rau caz j = i = 1). De la urmatoarele iteratii stim ca pe A[1] se afla un element <= decat x deci daca e cazul, acesta va opri indexul j sa ajunga pe pozitia j = 0 , iar pe A[n] se afla un element >= decat x deci i-ul nu poate depasi n.

2. the repeat-until loops stop on equal elements and swaps them. Why?

R:

- ➔ in felul acesta se stabilesc 'barierele' (ex.1) astfel incat indecsii sa ramana in intervalul [1,n]

De exemplu , daca am avea < si > atunci pentru exemplul 9,3,12,9,2,9,5 -> 9,3,5,9,2,9,12 -> i-ul depaseste n => eroare

- ➔ adaptare la cazul acesta: Hoare's update

3. First element pivot has an undesired worst case (leads O(n2) quicksort). Which is

it? Why is it undesired?

Pentru un sir sortat crescator:

pozitia lui i este gasita in timp constant, DAR pentru j trebuie ca sirul sa fie parcurs in intregime pana cand ajunge la pozitia 1 (algoritmul nu e adaptiv);

pentru :

QuickSort(1,n) ==> ~ n - 1 (aproximativ pentru ca sirul poate sa nu fie **strict** crescator)
QuickSort(2,n) ==> ~ n - 2
QuickSort(2,n) ==> ~ n - 3
.....
QuickSort(n-1,n) ==> ~ 1
_____(+)
= n(n-1)/2 comparatii --> O(n^2) doar partea de partition

4. using A[r] as pivot causes error execution. Why? Can it be avoided? Homework!

daca alegem ca pivot ultimul element din sirul dat A[r] algoritmul intra intr-o bucla infinita

cand ajunge intr-un subsir de 2 elemente deja ordonate

exemplu: 3,5,7 -> apelez quickSort , apelez partition(1,3) -> return 3 si revin in quickSort unde voi apela quickSort(A,left,partitionIndex) ~ (A,1,3) (apel identic cu cel initial → bucla infinita)

In QuickSort partitionIndex (rezultatul de la functia Partition) reprezinta elementul ce e cuprins intre left si right asftel ca atunci cand returnam indexul ultimului element (= right) din subsir , in quickSort left-ul si right-ul vor ramane neschimbate si reapelam functia partition cu aceeasi parametrii

La setarea pivotului pe prima pozitie nu apare problema aceasta pentru ca indexul returnat este primul element de la dreapta la stanga ce nu mai poate fi interschimbat => DECI partitionIndex != right => nu poate intra in bucla infinita

Cum am putea rezolva asta?

Obsevam ca problema apare atunci cand partitionIndex = right, deci in quickSort putem pune conditia ca atunci cand din partitionIndex = right sa apelam **quickSort(left, partitionIndex-1)**
quickSort(partitionIndex,right)

```
62
63  void quickSort(int a[], int left, int right)
64  {
65      if (left < right)
66      {
67          int partitionIndex = partition(a, left, right);
68          showArray(a, 1, n - 1);
69          if (partitionIndex != right )
70          {
71              quickSort(a, left, partitionIndex);
72              quickSort(a, partitionIndex + 1, right);
73
74          }
75          else
76          {
77              quickSort(a, left, partitionIndex - 1);
78              quickSort(a, partitionIndex , right);
79          }
80
81      }
82  }
83
```

SAU

Putem sa retinem un index ce creste doar atunci cand are se gaseste un element mai mic decat pivotul (algoritmul propus la SDA)

Partition(A,low,high)

1. Pivot = A[high]
2. s=low-1;
3. for I <- low,high-1
4. if(A[i] <= Pivot)
5. s<-s+1
6. swap(A[crt],A[s])
7. swap(A[s+1],A[high])
8. return s+1

5. using A[p] as pivot is essential in this implementation. Why? Homework!

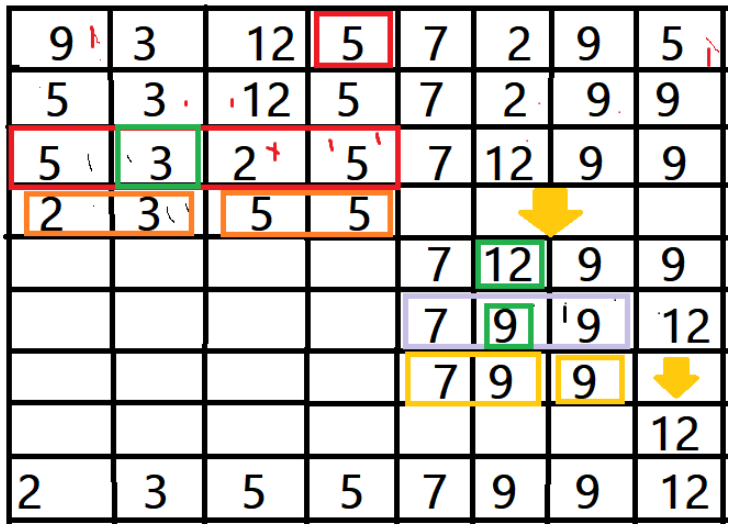
Algoritmul imparte sirul in 2 subsiruri pentru fiecare apel al functiei Partition: de la stanga la dreapta (pana la pivot) elementele mai mici decat acesta , iar restul elementele mai mari.

Alegererea pivotului ca primul element din array garanteaza faptul ca la fiecare apel recursiv vom alege ca pivot un element <= decat pivotul anterior (pentru fiecare "fiu" din spatiul de cautare al problemei)

Astfel daca valoarea pivotului e mica si in stanga lui se vor afla elemente si mai mici iar sirul se va sorta treptat de la stanga la dreapta

part2 Hoare Partition updated

trasare:



1.symmetric method. Works the same, whatever (middle, first, last) pivot is chosen.

2. the while loops stop on equal elements and swap them. Why? Why not allowing them in the partition they already belong and change the loops conditions to non-strict inequalities?

-> daca ajungem intr-un subsir de 2 elemente la stanga --> j <- 0

exemplu: 3,2 => pivot = 3 , i=1 --> a[1]=3 <= 3 ; i=2 --> a[2] = 2 <= 3 (se continua cautarea chiar daca a fost depasita lungimea array-ului)

-> alta eroare pivotul e chiar maximul / minimul => indexul i(sau j) parcurge tot subsirul pana la final (inceput) fara sa gaseasca un element > (<) decat pivotul → continua sa caute in afara intervalului [1,n] → error

-> in varianta Updated, dupa o parcurgere a sirului in partea dreapta a pivotului se vor alfa elementele <= si in partea dreapta cele >= (pastram egalitatea pentru ca poate gasim un element mai mic in partea dreapta pe care il putem interschimba cu un element = cu pivotul alfat in partea stanga → ceea ce e mai convenabil)

Deci pivotul poate fi vazut ca o bariera astfel incat indecsii sa ramana in [1,n]

3.In case i=j elements are swapped. It is redundant! Why to swap them? So can we change

if i<=j into if i<j? Any trap?

Cand i = j nu se va intra pe conditie => nu se executa operatiile i++ si j— → se va returna i = j

In quickSort se va apela din nou functia Partition pentru left = i si right = i (identic cu apelul aterior); astfel se intra intr-o bucla infinita

```
63  int  partitionUpdated(int arr[], int left, int right)
64  {
65      numarator = 0;
66      int pivot = arr[(left + right) / 2];
67      int i = left;
68      int j = right;
69
70      do
71      {
72          showArray(arr,1,n-1);
73          printf("%d %d\n", i, j);
74          while (arr[i] < pivot)
75          {
76              i++;
77          }
78          while (arr[j] > pivot)
79          {
80              j--;
81          }
82
83          if (i < j)
84          {
85              swapping(&arr[i], &arr[j]);
86              i++;
87              j--;
88          }
89
90
91      } while (!(j<i));
92
93      return i;
94  }
```

```
98  void quickSort(int a[], int left, int right)
99  {
100     if (left < right)
101     {
102         int partitionIndex = partitionUpdated(a, left, right);
103         showArray(a, 1, n - 1);
104         quickSort(a, left, partitionIndex - 1);
105         quickSort(a, partitionIndex  , right);
106     }
107 }
```