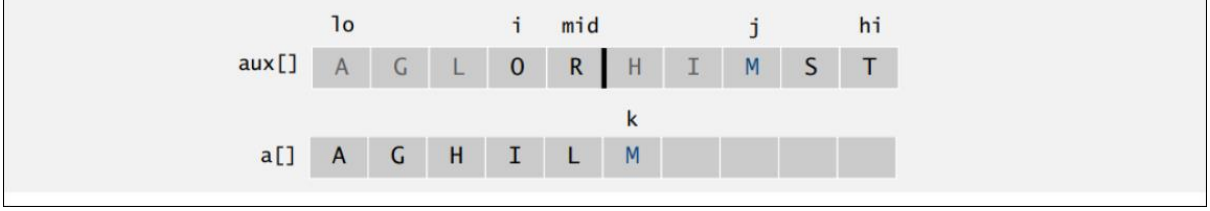
ALGORITMAT E RENDITJES

MERGE Sort

- Bashkimi i gjysmave të renditura në një vektor të renditur
- Ka nevojë per memorie shtese

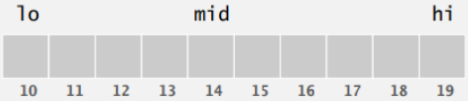
Implementimi në Java

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
   assert isSorted(a, lo, mid);    // precondition: a[lo..mid]   sorted
   assert isSorted(a, mid+1, hi);  // precondition: a[mid+1..hi] sorted

   for (int k = lo; k <= hi; k++)                                          copy
      aux[k] = a[k];

   int i = lo, j = mid+1;
   for (int k = lo; k <= hi; k++)                                          merge
   {
      if      (i > mid)              a[k] = aux[j++];
      else if (j > hi)               a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                           a[k] = aux[i++];
   }

   assert isSorted(a, lo, hi);     // postcondition: a[lo..hi] sorted
}
```

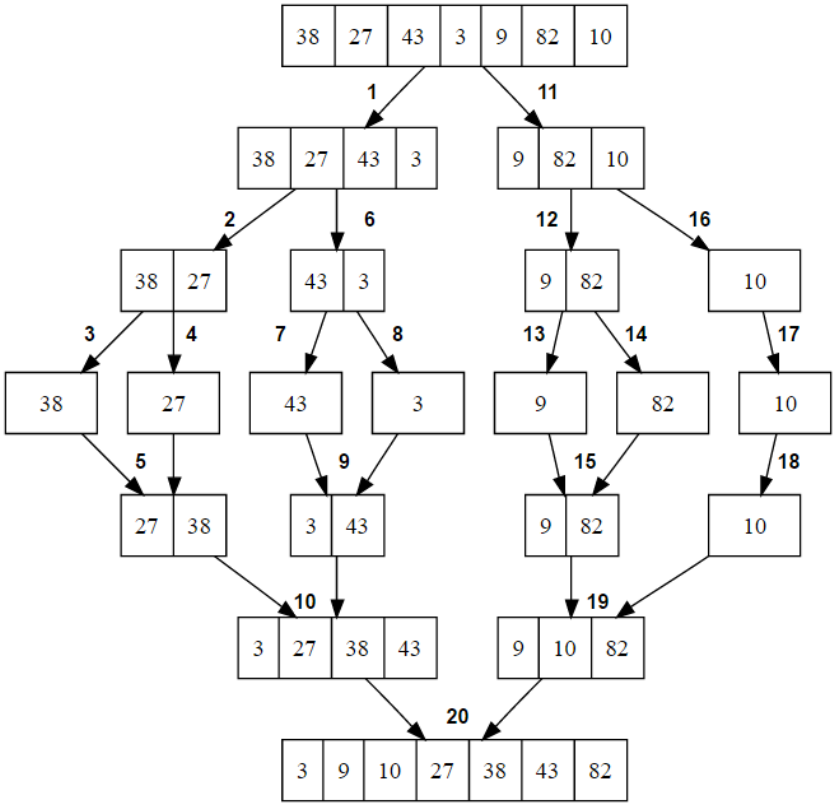


```
public class Merge
{
   private static void merge(...)
   {  /* as before */  }

   private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
   {
      if (hi <= lo) return;
      int mid = lo + (hi - lo) / 2;
      sort(a, aux, lo, mid);
      sort(a, aux, mid+1, hi);
      merge(a, aux, lo, mid, hi);
   }

   public static void sort(Comparable[] a)
   {
      aux = new Comparable[a.length];
      sort(a, aux, 0, a.length - 1);
   }
}
```

Matrica ndihmëse (aux) duhet të krijohet vetëm një herë edhe pastaj kalohet si argument në thirrjet rekursive! PSE?



Shembull të iteracioneve për të rradhitur një vektor në rend rritës



Analiza e algoritmit

Kompleksiteti në kohë:

Rasti më i mirë: O(n*logn)

Rasti mesatar: O(n*logn)

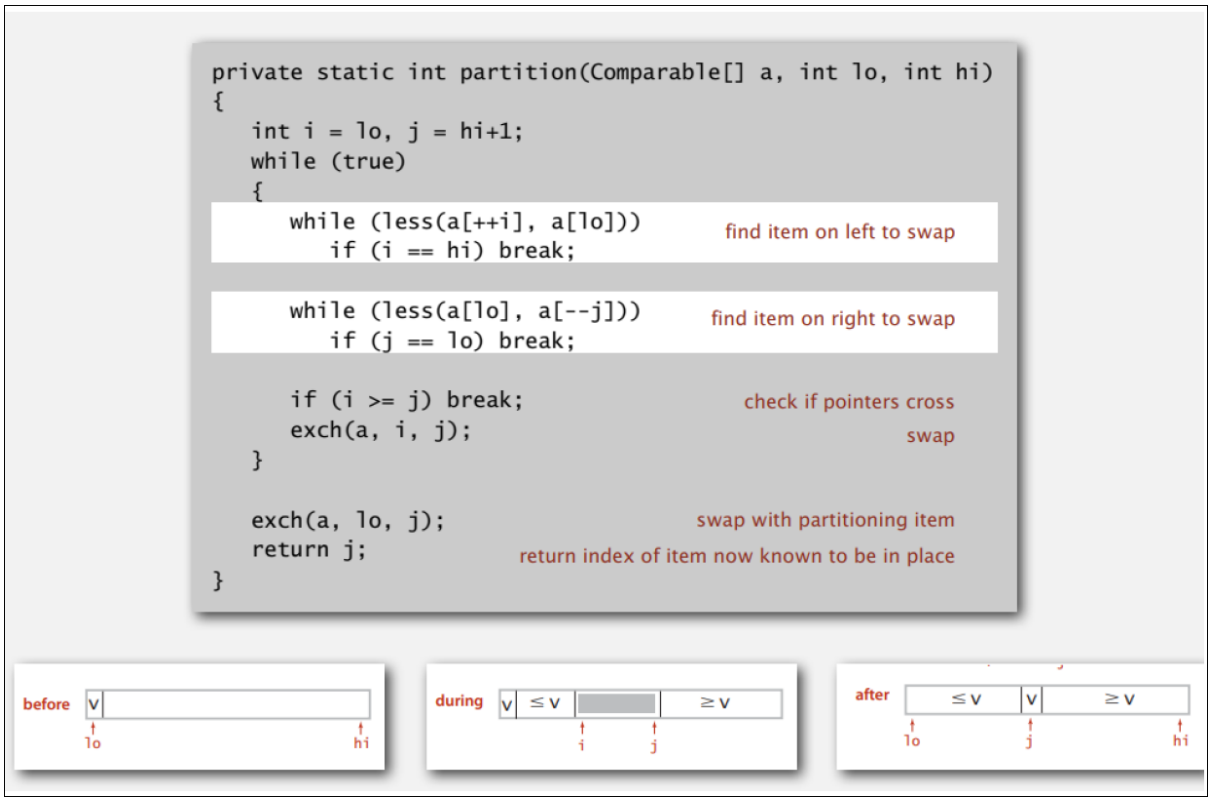
Rasti më i keq : O(n*logn)

Kompleksiteti në hapësirë(call stack): O(n)

Quick Sort

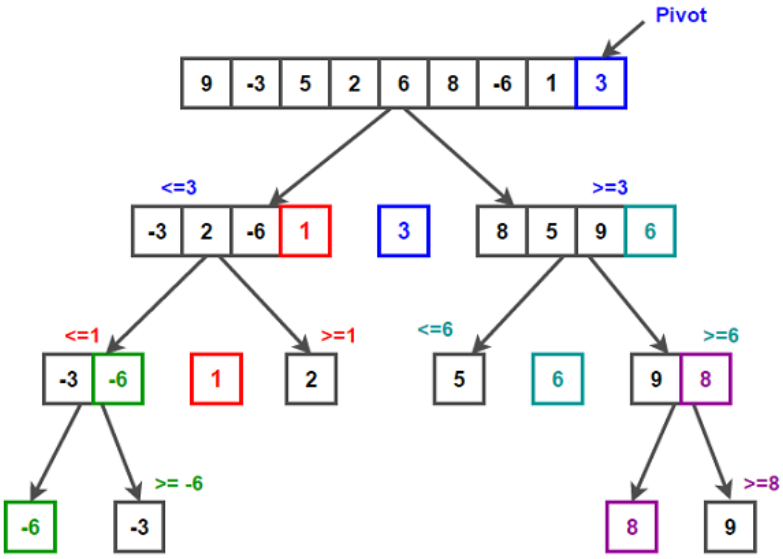
- Algoritëm in-place sorting
- Përdor një element të quajtur si pivot për të renditur elementët
- Një nga algoritmet më të mira për rastin mesatar (average)

Implementimi në Java



```
public class Quick
{
   private static int partition(Comparable[] a, int lo, int hi)
   {  /* see previous slide */  }

   public static void sort(Comparable[] a)
   {
      StdRandom.shuffle(a);
      sort(a, 0, a.length - 1);
   }

   private static void sort(Comparable[] a, int lo, int hi)
   {
      if (hi <= lo) return;
      int j = partition(a, lo, hi);
      sort(a, lo, j-1);
      sort(a, j+1, hi);
  }
}
```

Shembull të iteracioneve për të rradhitur një vektor në rend rritës



Analiza e algoritmit

Kompleksiteti në kohë:

Rasti më i mirë: O(n*logn)

Rasti mesatar: O(n*logn)

Rasti më i keq (pivot elementi më i vogël ose më i madh): $O(n^2)$

Kompleksiteti në hapësirë (stack): O(n)

Ushtrime

1. Cilat algoritma renditje përdorin metodën divide and conquer për të renditur një vektor?
2. Cilat algoritme renditje kërkojnë hapësirë shtesë?
3. Cila nga algoritmat e renditjes është aplikuar më poshtë? Argumentoni përgjigjen tuaj.

83 5 8 12 65 72 71
5 83 8 12 65 72 71
5 8 83 12 65 72 71
5 8 12 83 65 72 71
5 8 12 65 83 72 71
5 8 12 65 72 83 71
5 8 12 65 71 72 83

- Selection Sort
- Insertion Sort
- Merge Sort
- None of the above

4. Cili prej algoritmave të renditjes (Insertion Sort, Selection Sort, Merge Sort) do të zgjidhnit të përdornit për:
   a) Vektorë të vegjël në hapësirë memorie të limituar.
   b) Vektorë që janë pothuajse të renditur.
   c) Vektorë të mëdhenj me vlera numerike të rastësishme.

5. Tregoni hap pas hapi renditjen e vektorit A = (13, 17, 24, 11, 91, 3, 10, 21, 100) në rend rritës duke përdorur Quick Sort. Diskutoni rreth kompleksitetit në kohë dhe në hapesirë.

6. Ndërtoni hap pas hapi të gjithë iteracionet që do të ndiqen për të rradhitur vektorin e mëposhtëm:
   int a[] = {44, 88, 55, 99, 66, 33, 22, 88, 77}
   a. Quick sort
   b. Merge sort

7. Gjatë përzgjedhjes së vlerës pivot, një mënyrë e thjeshtë është përzgjedhja e e një pozicioni specifik në çdo pjesë. Nëse përdoret kjo metodikë, a ka rëndësi nëse zgjedhim vlerë në fillim, në fund apo në mes të çdo nëndarje?
   a. Zgjedhja më e mirë është pozicioni i parë
   b. Zgjedhja më e mirë është pozicioni i fundit
   c. Zgjedhja më e mirë është pozicioni i mesit
   d. Nuk ka rëndësi pozicioni

8. Cili është rasti mesatar (average) për QuickSort për të rradhitur një vektor me n elemente?
   a. O(logn)
   b. O(n)
   c. $O(n^2)$
   d. O(n*logn)

9. Cili është kompleksiteti në rastin më të keq (worst case) për algoritmin QuickSort për të rradhitur një vektor me n elemente?
   a. O(logn)
   b. O(n)
   c. $O(n^2)$
   d. O(n*logn)

10. Në QuickSort cilat kompleksitete në kohë janë të barabarta?
    a. Worst dhe Average
    b. Worst dhe Best
    c. Worst, Average, Best
    d. Best dhe Average

11. Duke marrë në konsideratë metodat top-down mergesort dhe bottom-up mergesort rradhitni vektorin e mëposhtëm:
    E A S Y Q U E S T I O N

12. Çfarë ndodh nëse nëse në algoritmin mergesort shtojmë një kusht if a[mid] është më i vogël ose i barabartë me a[mid+1]. Sa do të ishte kompleksiteti i mergesort nëse plotësimi i këtij kushti sjell mosthirrjen e metodës merge()?

13. **Kendall tau distance:** Një permutation është një vektor me N integer ku secili prej këtyre integer me vlerat nga 0 në N-1 gjendet vetëm një here. Supozoni se kemi dy rradhitje te ndryshme. P.sh.: Rradhitja 1: 0,3,1,6,2,5,4 dhe rradhitja 2: 1,0,3,6,4,2. Distanca Kendall tau në këtë rast do të ishte numri i çifteve te cilat ndodhen ne rradhitje te ndryshme per dy rastet e dhëna. Në këtë rast distanca Kendall tau do të ishte katër, duke qenë se kemi katër çifte të cilat ndodhen në rradhitje të kundërt: 0-1, 3-1, 2-4 dhe 5-4.
    Krijoni një program i cili llogarit distancën e njohur si Kendall Tau:
    a. Me rend n^2
    b. Me rend nlogn.

14. Rreth sa krahasime bën algoritmi quick sort, në momentin që jepet një vektor me N elementë të njëjtë?

15. Jepen dy vektorë të rradhitur a[] dhe b[], me përmasat n1 dhe n2. Ndërtoni një algoritëm për të gjetur elementin e k-të më të madh. Rendi në rastin më të keq për këtë algoritëm duhet të jetë logn, ku n=n1+n2.

16. Veprimi merge tek algoritmi Mergesort tani kërkon një kohë konstante. Sa është kompleksiteti në kohë (Big O) në rastin më të keq?
17. Për secilin nga algoritmet e mëposhtme të renditjes, tregoni sa eshte θ (si funksion i n-së) i numrit të krahasimeve që duhen për të renditur në rend rritës një matricë me n elemente të ndryshme edhe qe eshte e renditur në rend zbritës.
    a. MergeSort
    b. QuickSort

18. Kodi i mëposhtëm llogarit numrin e elementeve të përbashkët të dy matricave a edhe b. Supozojmë se asnjë matricë nuk ka elemente që përsëriten. Sa është kompleksiteti në kohë (Big O)? Shpjegoni qartë përgjigjen.

```
int intersection(int[] a, int[] b){
    mergesort(b);
    int intersect = 0;
    for (int x : a) {
        if (binarySearch(b, x) >= 0) {
            intersect++;
        }
    }
    return intersect;
}
```

19. Quicksort është një algoritëm rekursiv edhe in-place renditjeje i cili përdor veprimin ***partititon*** mbi matricën që duhet të renditet edhe më pas thërret veten 2 herë për renditjen e dy pjesëve te perftuara pas partition.
    a. Tregoni qëllimin edhe parametrat që merr is input proçedura partition. Shpjegoni çfarë është pivot.
    b. Jepni kodin e funksionit Quicksort i cili thërret funksionin partititon (pa kodin e partition).
    c. Alnalizoni sjelljen në rastin më të keq për QuickSort edhe tregoni çfarë mund të bëjmë për të shmangur këtë rast.

20. Konsideroni një element në një matricë që do renditet me algoritmin MergeSort. Tregoni sa herë do të lëvizë ky element në një pozicion tjetër gjatë ekzekutimit të algoritmit? Shpjegoni përgjigjen.

21. Ushtrim implementimi i Merge Sort
    https://opendsa-server.cs.vt.edu/OpenDSA/Books/Everything/html/Mergesort.html
22. Ushtrim implementimi i Quick Sort
    https://opendsa-server.cs.vt.edu/OpenDSA/Books/Everything/html/Quicksort.html