Curs Algoritmi Fundamentali
Alte sortări: Sortarea rapidă (QuickSort) şi Bucket Sort

Universitatea *Transilvania* din Braşov
Facultatea de Matematică şi Informatică

2020

Contents

QuickSort - Generalităţi

Ideea algoritmului de sortare rapidă - Quick Sort:

- Se rearanjează şi apoi se împarte tabloul unidimensional, de dimensiune *n*, arr[1..n] în două subtablouri arr[1..m] şi arr[m+1..n] astfel încât componentele lui arr[1..m] sunt mai mici decât arr[m+1..n]; este importantă o rearanjare anterioară, pentru a evita diverse probleme legate de complexitate

QuickSort - Generalităţi

Ideea algoritmului de sortare rapidă - Quick Sort:

- Se rearanjează şi apoi se împarte tabloul unidimensional, de dimensiune *n*, arr[1..n] în două subtablouri arr[1..m] şi arr[m+1..n] astfel încât componentele lui arr[1..m] sunt mai mici decât arr[m+1..n]; este importantă o rearanjare anterioară, pentru a evita diverse probleme legate de complexitate
- Se sortează fiecare dintre subtablourile obţinute aplicând exact aceeaşi strategie de împărţire

QuickSort - Generalităţi

Ideea algoritmului de sortare rapidă - Quick Sort:

- Se rearanjează şi apoi se împarte tabloul unidimensional, de dimensiune *n*, arr[1..n] în două subtablouri arr[1..m] şi arr[m+1..n] astfel încât componentele lui arr[1..m] sunt mai mici decât arr[m+1..n]; este importantă o rearanjare anterioară, pentru a evita diverse probleme legate de complexitate
- Se sortează fiecare dintre subtablourile obţinute aplicând exact aceeaşi strategie de împărţire
- Se concatenează subtablourile sortate

QuickSort - Generalităţi

Algoritmul Quick Sort este un algoritm de sortare prin pivotaj. Astfel, subliniem câteva idei pentru construirea/alegerea unui pivot:

- Se alege o valoare aleatore din tablou (prima, ultima sau una arbitrară), aceasta reprezentând valoarea pivotului

QuickSort - Generalităţi

Algoritmul Quick Sort este un algoritm de sortare prin pivotaj. Astfel, subliniem câteva idei pentru construirea/alegerea unui pivot:

- Se alege o valoare aleatore din tablou (prima, ultima sau una arbitrară), aceasta reprezentând valoarea pivotului
- Se rearanjează elementele tabloului astfel încât toate elementele care sunt mai mici decât valoarea aleasă să se afle în prima parte (stânga) a tabloului, iar valorile mai mari decât pivotul să se afle în partea a doua (dreapta) a tabloului

QuickSort - Generalităţi

Algoritmul Quick Sort este un algoritm de sortare prin pivotaj. Astfel, subliniem câteva idei pentru construirea/alegerea unui pivot:

- Se alege o valoare aleatore din tablou (prima, ultima sau una arbitrară), aceasta reprezentând valoarea pivotului
- Se rearanjează elementele tabloului astfel încât toate elementele care sunt mai mici decât valoarea aleasă să se afle în prima parte (stânga) a tabloului, iar valorile mai mari decât pivotul să se afle în partea a doua (dreapta) a tabloului
- Se pune valoarea pivotului pe poziţia sa finală (astfel încât în stânga să avem elementele mai mici decât pivotul, iar în dreapta, elementele mai mari ca valoarea pivotului)

QuickSort - Generalităţi

Un bun pivot împarte tabloul curent în două subtablouri de dimensiuni apropiate (partiţionare echilibrată). Alegerea unui bun pivot este direct proporţională cu obţinerea unei complexităti mai bune.
O idee de rearanjare a elementelor:

- Se folosesc doi indici: unul care porneşte de la poziţia primul element (limităInferioară) iar celălalt care porneşte de la poziţia ultimului element (limităSuperioară)

QuickSort - Generalităţi

Un bun pivot împarte tabloul curent în două subtablouri de dimensiuni apropiate (partiţionare echilibrată). Alegerea unui bun pivot este direct proporţională cu obţinerea unei complexităti mai bune.
O idee de rearanjare a elementelor:

- Se folosesc doi indici: unul care porneşte de la poziţia primul element (limităInferioară) iar celălalt care porneşte de la poziţia ultimului element (limităSuperioară)
- Se incrementează/decrementează indicii până când se identifică o inversiune (o pereche de indici $i < j$ cu proprietatea că arr[i] $>$ pivot şi arr[j] $<$ pivot)

QuickSort - Generalităţi

Un bun pivot împarte tabloul curent în două subtablouri de dimensiuni apropiate (partiţionare echilibrată). Alegerea unui bun pivot este direct proporţională cu obţinerea unei complexităti mai bune.
O idee de rearanjare a elementelor:

- Se folosesc doi indici: unul care porneşte de la poziţia primul element (limităInferioară) iar celălalt care porneşte de la poziţia ultimului element (limităSuperioară)
- Se incrementează/decrementează indicii până când se identifică o inversiune (o pereche de indici $i < j$ cu proprietatea că arr[i] $>$ pivot şi arr[j] $<$ pivot)
- Se repară situţia inversiunii prin interschimbarea elementelor

QuickSort - Generalităţi

Un bun pivot împarte tabloul curent în două subtablouri de dimensiuni apropiate (partiţionare echilibrată). Alegerea unui bun pivot este direct proporţională cu obţinerea unei complexităti mai bune.
O idee de rearanjare a elementelor:

- Se folosesc doi indici: unul care porneşte de la poziţia primul element (limităInferioară) iar celălalt care porneşte de la poziţia ultimului element (limităSuperioară)
- Se incrementează/decrementează indicii până când se identifică o inversiune (o pereche de indici $i < j$ cu proprietatea că arr[i] $>$ pivot şi arr[j] $<$ pivot)
- Se repară situţia inversiunii prin interschimbarea elementelor
- Se continuă procesul până când indicii se "suprapun"

QuickSort - Pseudocod

```
procedure Partiţionare   {Partiţionează tabloul a[s..d]}
*selectează elementul x (de regulă de la mijlocul
                         intervalului de partiţionat)
  repetă
    *caută primul element a[i]>x, parcurgând
       intervalul de la stânga la dreapta
    *caută primul element a[j]<x, parcurgând
       intervalul de la dreapta la stânga
    dacă i<=j  atunci
      *interschimbă pe a[i] cu a[j]

  până când  parcurgerile se întâlnesc (i>j)

procedure QuickSort(s,d);
  *partiţionează intervalul s,d faţă de Mijloc
  dacă există partiţie stânga atunci
    QuickSort(s,Mijloc-1)
  dacă există partiţie dreapta atunci
    QuickSort(Mijloc+1,d);
```

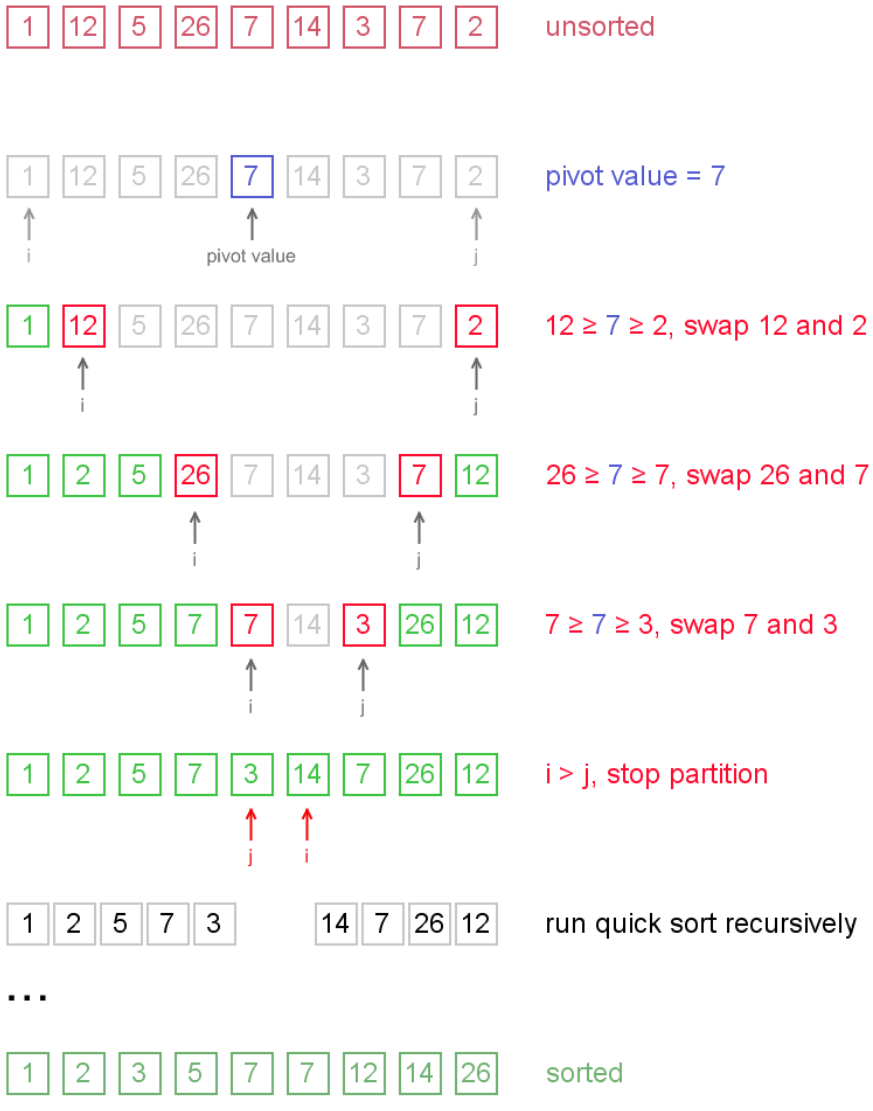
QuickSort - Implementare C++

```
void QuickSort(int arr[], int left, int right)
{
    int i = left, j = right;
    int tmp;
    int pivot = arr[(left + right) / 2];
    /*Partitionare */
    while (i <= j) {
        while (arr[i] < pivot)
            i++;
        while (arr[j] > pivot)
            j--;
        if (i <= j) {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
            i++;
            j--;
        }
    }

    /* Recursivitate */

    if (left < j)
        Quick Sort(arr, left, j);
    if (i < right)
        Quick Sort(arr, i, right);
}
```

QuickSort - Exemplu



Performanţa algoritmului QuickSort

Timpul de rulare al acestui algoritm depinde de partiţionare: dacă este sau nu realizată într-un mod balansat. Dacă partiţionarea este una balansată, algoritmul are timp de rulare $\Theta(n * \log n)$, în caz contrar ordinul de timp fiind unul pătratic.

1. Cel mai defavorabil caz: procedura de partiţionare produce o regiune cu $n - 1$ elemente şi alta cu 1 element

Performanţa algoritmului QuickSort

Timpul de rulare al acestui algoritm depinde de partiţionare: dacă este sau nu realizată într-un mod balansat. Dacă partiţionarea este una balansată, algoritmul are timp de rulare $\Theta(n * \log n)$, în caz contrar ordinul de timp fiind unul pătratic.

1. Cel mai defavorabil caz: procedura de partiţionare produce o regiune cu $n - 1$ elemente şi alta cu 1 element
2. Cel mai favorabil caz: procedura de partiţionare ar produce două subşiruri de mărime $\frac{n}{2}$

Performanţa algoritmului QuickSort

Timpul de rulare al acestui algoritm depinde de partiţionare: dacă este sau nu realizată într-un mod balansat. Dacă partiţionarea este una balansată, algoritmul are timp de rulare $\Theta(n * \log n)$, în caz contrar ordinul de timp fiind unul pătratic.

1. Cel mai defavorabil caz: procedura de partiţionare produce o regiune cu $n - 1$ elemente şi alta cu 1 element
2. Cel mai favorabil caz: procedura de partiţionare ar produce două subşiruri de mărime $\frac{n}{2}$
3. Cazul mediu: nici perfect balansate şi nici total nebalansate

QuickSort - Cel mai defavorabil caz

Partiţionarea va avea un timp liniar $\Theta(n)$ şi $T(1) = \Theta(1)$, astfel că formula de recurenţă pentru timpul total este: $T(n) = T(n-1) + \Theta(n)$. Pentru a evalua această recurenţă utilizăm iteraţiile:

$$T(n) = T(n-1) + \Theta(n)$$

$$T(n) = T(n-2) + \Theta(n-1) + \Theta(n)$$

$$\dots$$

Astfel:

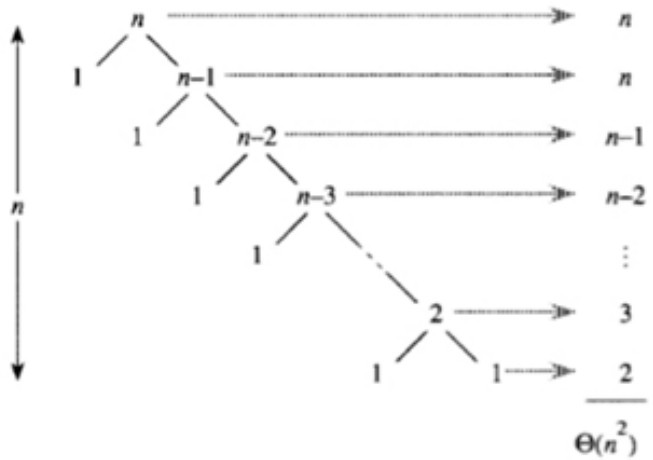
$$T(n) = \sum_{k=1}^{n} \Theta(n)$$

$$T(n) = \Theta(\sum_{k=1}^{n} n)$$

$$T(n) = \Theta(n^2)$$

QuickSort - Cel mai defavorabil caz

Arborele de recursivitate pentru care partiţia separă şirul în 1, respectiv $n - 1$ elemente.



QuickSort - Cel mai favorabil caz

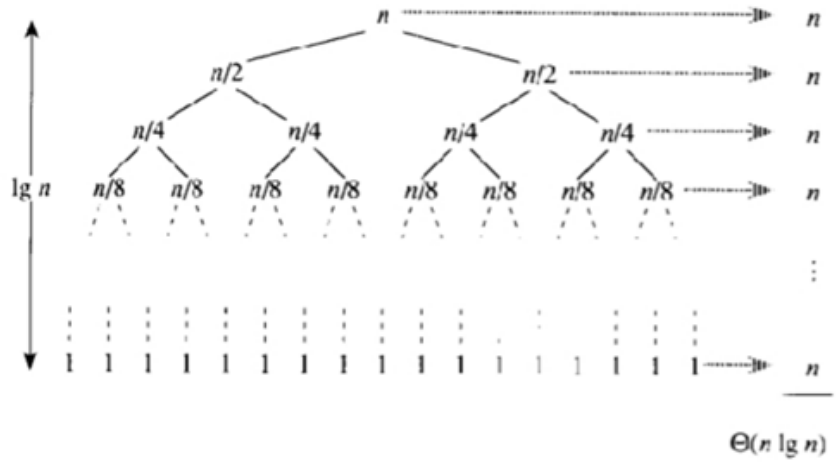
Recurenţa este dată de:

$$T(n) = 2 * T(\frac{n}{2}) + \Theta(n)$$

ce are ca soluţie

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

.



QuickSort - Cazul mediu

Să presupunem că procedura de partiţionare produce o repartizare de 90% elemente într-o parte, 10% în cealaltă, dintr-un total de 100% elemente. Vom obţine o recurenţă:

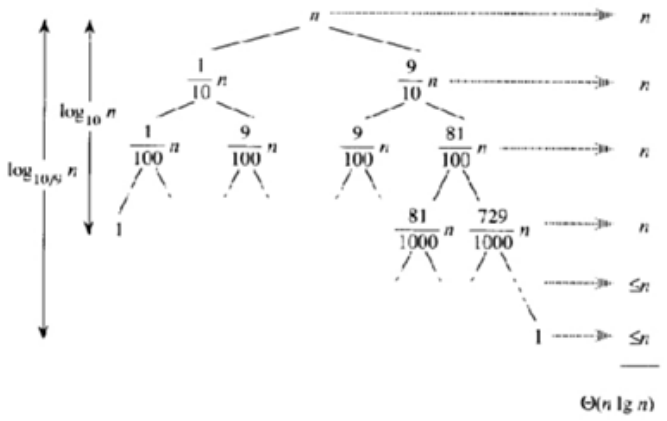
$$T(n) = T(\frac{9n}{10}) + T(\frac{n}{10}) + \Theta(n)$$

$$T(n) = T(\frac{9n}{10}) + T(\frac{n}{10}) + n$$

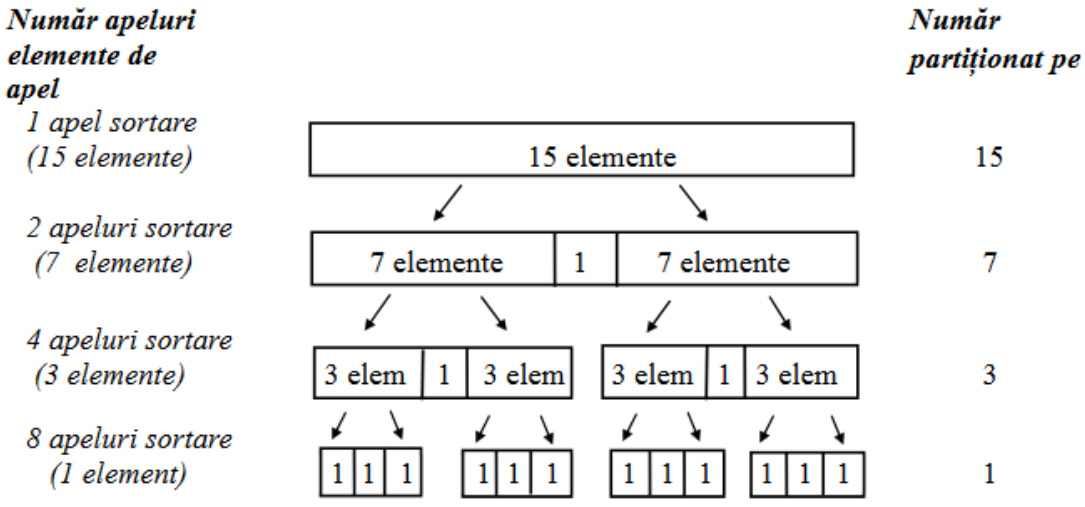
Fiecare nivel de recurenţă are un cost total de n, până când condiţia limită este îndeplinită pentru o adâncime de $\log_1 0n = \Theta(\log n)$. Astfel costul total este $\Theta(n * \log n)$, deci algoritmul este tot în timp $n * \log n$ (baza diferă)

QuickSort - Cazul mediu

Arborele de recursivitate în care partiţia prouduce întotdeauna o repartizare de 9 la 1.



QuickSort - Exemplu



$$\log_2^{15} \approx 4$$

4 treceri prin toate elementele tabloului sau 4 paşi de partiţionare integrală a tabloului

Dezavantaje?

QuickSort - Avantaje şi dezavantaje

- (+) necesită, în cazul mediu, $O(\log_2 n)$ memorie suplimentară, mult mai redusă decât sortarea prin interclasare.

QuickSort - Avantaje şi dezavantaje

- (+) necesită, în cazul mediu, $O(\log_2 n)$ memorie suplimentară, mult mai redusă decât sortarea prin interclasare.
- (+) cea mai bună alegere când viteza este foarte importantă, indiferent de dimensiunea setului de date

QuickSort - Avantaje şi dezavantaje

- (+) necesită, în cazul mediu, $O(\log_2 n)$ memorie suplimentară, mult mai redusă decât sortarea prin interclasare.
- (+) cea mai bună alegere când viteza este foarte importantă, indiferent de dimensiunea setului de date
- (-) poate duce la umplerea stivei când se utilizează seturi largi de date

QuickSort - Avantaje şi dezavantaje

- (+) necesită, în cazul mediu, $O(\log_2 n)$ memorie suplimentară, mult mai redusă decât sortarea prin interclasare.
- (+) cea mai bună alegere când viteza este foarte importantă, indiferent de dimensiunea setului de date
- (-) poate duce la umplerea stivei când se utilizează seturi largi de date
- (-) în cazul cel mai defavorabil performanţa metodei scade catastrofal (când la fiecare partiţionare este selectată cea mai mare (cea mai mică) valoare ca şi pivot, dar şi când avem duplicate)

QuickSort - Avantaje şi dezavantaje

- (+) necesită, în cazul mediu, $O(\log_2 n)$ memorie suplimentară, mult mai redusă decât sortarea prin interclasare.
- (+) cea mai bună alegere când viteza este foarte importantă, indiferent de dimensiunea setului de date
- (-) poate duce la umplerea stivei când se utilizează seturi largi de date
- (-) în cazul cel mai defavorabil performanţa metodei scade catastrofal (când la fiecare partiţionare este selectată cea mai mare (cea mai mică) valoare ca şi pivot, dar şi când avem duplicate)
- (-) algoritmul de sortare rapidă este instabil

QuickSort - Avantaje şi dezavantaje

- (+) necesită, în cazul mediu, $O(\log_2 n)$ memorie suplimentară, mult mai redusă decât sortarea prin interclasare.
- (+) cea mai bună alegere când viteza este foarte importantă, indiferent de dimensiunea setului de date
- (-) poate duce la umplerea stivei când se utilizează seturi largi de date
- (-) în cazul cel mai defavorabil performanţa metodei scade catastrofal (când la fiecare partiţionare este selectată cea mai mare (cea mai mică) valoare ca şi pivot, dar şi când avem duplicate)
- (-) algoritmul de sortare rapidă este instabil
- (-) are performanţe slabe în cazul sortărilor banale, atunci când operează asupra unor liste aproape ordonate.

QuickSort - Avantaje şi dezavantaje

- (+) necesită, în cazul mediu, $O(\log_2 n)$ memorie suplimentară, mult mai redusă decât sortarea prin interclasare.
- (+) cea mai bună alegere când viteza este foarte importantă, indiferent de dimensiunea setului de date
- (-) poate duce la umplerea stivei când se utilizează seturi largi de date
- (-) în cazul cel mai defavorabil performanţa metodei scade catastrofal (când la fiecare partiţionare este selectată cea mai mare (cea mai mică) valoare ca şi pivot, dar şi când avem duplicate)
- (-) algoritmul de sortare rapidă este instabil
- (-) are performanţe slabe în cazul sortărilor banale, atunci când operează asupra unor liste aproape ordonate.
- (+/-) are performanţe deosebite în cazul tablourilor dezordonate

Metode de sortare. Analiză timp.

Viteza de sortare pentru o multime de 10.000.000 de elemente

	Sir initial aleator	Sir gata sortat	Sir sortat invers
Selection	-	-	-
Insertion	-	2.130s	-
HeapSort (C++)	17.901s	14.215s	14.330s
QuickSort	4.807s	-	-
QuickSort (C)	4.357s	2.826s	2.734s
STL Sort (C++)	5.961s	3.82s	3.340s

Bucket Sort (Bin Sort) - O altfel de sortare

- Este un algoritm de sortare util pentru situaţia de numere pozitive întregi care nu depăşesc o anumită limită superioară.

Bucket Sort (Bin Sort) - O altfel de sortare

- Este un algoritm de sortare util pentru situaţia de numere pozitive întregi care nu depăşesc o anumită limită superioară.
- Cel mai bun caz de a utiliza acest algoritm: avem multe valori de sortat care se repetă/sunt uniform distribuite peste un rang dat.

Bucket Sort. Exemple (I)

Avem un tablou cu lunile de naştere pentru mai multe persoane, dorim să ordonăm acest tablou.

****

Sortaţi un tablou mare de numere între 0.0 şi 1.0 uniform distribuite. Cum sortăm numerele eficient?

Bucket Sort. Exemple (II)

Să considerăm tabloul: [12, 7, 3, 5, 12, 5, 4, 2, 8, 6, 9, 1, 2, 5, 2, 1, 8, 10, 8, 1, 4, 12]
Tablou auxiliar cu 12 elemente [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Parcurgem tabloul de sortat
Elementul 12 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
Elementul 7 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]
Elementul 3 [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1]
Elementul 5 [0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1]
Elementul 12 [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 2]
...
La final: [3, 3, 1, 2, 3, 1, 1, 3, 1, 1, 0, 3]
Pentru a construi tabloul sortat, parcurgem tabloul auxiliar și punem 3 de 1, 3 de 2, etc.
Tabloul sortat: [1, 1, 1, 2, 2, 2, 3, 4, 4, 5, 5, 5, 6, 7, 8, 8, 8, 9, 10, 12, 12, 12]

Bucket Sort. Pseudocod

```
subalgoritm BucketSort(tablou, n, max) este:
//tablou - tabloul de ordonat, n - lungimea, max - valoarea maximă care apare
în tablou
    taux:Întreg[max]
    pentru i ← 1, n, 1 execută
        taux[tablou[i]] ← taux[tablou[i]] + 1
    sf_pentru
    index ← 1
    pentru i ← 1, max, 1 execută
        pentru j ← 1, taux[i], 1 execută
            tablou[index] ← i
            index ← index + 1
        sf_pentru
    sf_pentru
sf_subalgoritm
```

Bucket Sort. Complexitate

- Complexitatea pentru BucketSort este: $\Theta(n + max)$ - Discuţii

Bucket Sort. Complexitate

- Complexitatea pentru BucketSort este: $\Theta(n + max)$ - Discuţii
- Ce se întâmplă dacă avem numere negative uniform distribuite şi vrem să se sortăm cu BucketSort? Cum facem?

Bucket Sort. More about

- http://codercorner.com/RadixSortRevisited.htm

Bucket Sort. More about

- http://codercorner.com/RadixSortRevisited.htm
- https://www.geeksforgeeks.org/bucket-sort-to-sort-an-array-with-negative-numb

Tipuri de sortări diverse. Când folosim/Ce folosim?

- Metoda selecţiei: + cel mai intuitiv mod de sortare, - chiar şi cel mai perfect sortat input, necesită minim o parcurgere, + se foloseşte când faci ceva rapid şi "dirty", + in-place

Tipuri de sortări diverse. Când folosim/Ce folosim?

- Metoda selecţiei: + cel mai intuitiv mod de sortare, - chiar şi cel mai perfect sortat input, necesită minim o parcurgere, + se foloseşte când faci ceva rapid şi "dirty", + in-place
- Metoda bulelor: + util pentru seturi mici de date, + uşor de implementat, + in-place, - eficienţă redusă

Tipuri de sortări diverse. Când folosim/Ce folosim?

- Metoda selecţiei: + cel mai intuitiv mod de sortare, - chiar şi cel mai perfect sortat input, necesită minim o parcurgere, + se foloseşte când faci ceva rapid şi "dirty", + in-place
- Metoda bulelor: + util pentru seturi mici de date, + uşor de implementat, + in-place, - eficienţă redusă
- Sortare Interclasare: + poate fi folosit pentru date de orice mărime, + stable, - nu e in-place, + se foloseşte intens pentru sortarea de linked list, unde accesul secvenţial este necesar

Tipuri de sortări diverse. Când folosim/Ce folosim?

- QuickSort: + rapid în cazuri de date random, + in-place, chiar daca e recursiv, + rapid (ciclul repetititv din interior este foarte scurt şi se poate optimiza foarte bine), + are performanţe deosebite în cazul tablourilor dezordonate, - foarte nesigur pentru cazul defavorabil ($O(n^2)$), - performanţe slabe pentru tablourile banale

Tipuri de sortări diverse. Când folosim/Ce folosim?

- QuickSort: + rapid în cazuri de date random, + in-place, chiar daca e recursiv, + rapid (ciclul repetititv din interior este foarte scurt şi se poate optimiza foarte bine), + are performanţe deosebite în cazul tablourilor dezordonate, - foarte nesigur pentru cazul defavorabil ($O(n^2)$), - performanţe slabe pentru tablourile banale
- BucketSort: + poate fi folosit ca şi un algoritm "extern" de sortare, + rezultate foarte bune atunci când datele sunt uniform distribuite, - complexitatea creşte foarte mult atunci când mărim numărul de inputuri

Tipuri de sortări diverse. Când folosim/Ce folosim?

- QuickSort: + rapid în cazuri de date random, + in-place, chiar daca e recursiv, + rapid (ciclul repetititv din interior este foarte scurt şi se poate optimiza foarte bine), + are performanţe deosebite în cazul tablourilor dezordonate, - foarte nesigur pentru cazul defavorabil ($O(n^2)$), - performanţe slabe pentru tablourile banale
- BucketSort: + poate fi folosit ca şi un algoritm "extern" de sortare, + rezultate foarte bune atunci când datele sunt uniform distribuite, - complexitatea creşte foarte mult atunci când mărim numărul de inputuri
- `https://www.toptal.com/developers/sorting-algorithms/`

Ce (criterii) contează în alegerea unui algoritm de sortare?

- Cantitatea de date pe care dorim să le sortăm

Ce (criterii) contează în alegerea unui algoritm de sortare?

- Cantitatea de date pe care dorim să le sortăm
- Cât de sortate sunt deja datele

Ce (criterii) contează în alegerea unui algoritm de sortare?

- Cantitatea de date pe care dorim să le sortăm
- Cât de sortate sunt deja datele
- Complexitatea de timp (diferă de running time)

Ce (criterii) contează în alegerea unui algoritm de sortare?

- Cantitatea de date pe care dorim să le sortăm
- Cât de sortate sunt deja datele
- Complexitatea de timp (diferă de running time)
- Complexitatea de spaţiu

Aplicaţie 1
Hărţi mentale (Mind mapping)

Obiective:
- Folosirea hărţilor mentale pentru sintetizarea şi structurarea informaţiei;
- Stimularea creativităţii

Repartizare în timp: 100 minute

Exerciţiu introductiv: În 5 minute vă rog să notaţi idei-notiţe pentru un discurs despre gândire creativă şi informatică pe care urmează să îl ţineţi în faţa unui public.

Introducerea temei: Ce este o hartă mentală
Timp alocat: 50 min

Mind map este conceput în jurul unui singur concept central reprezentat printr-o imagine și/sau un cuvânt la care se conectează, prin intermediul link-urilor/legăturilor, alte concepte importante, de care, la rândul lor sunt conectate ramuri cu alte concepte. Toate aceste concepte formează o structură radială sau de foc de artificii.

Alcătuirea Mind map:
- Nod central;
- Legături etichetate;
- Ramuri cu sub-noduri;

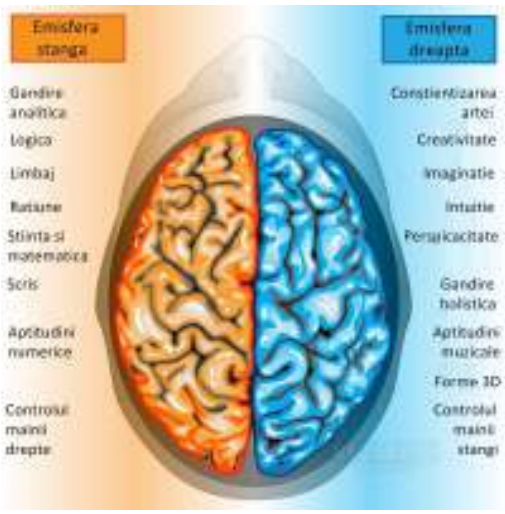
Un Mind map stimulează mai multe zone ale creierului (care ajută memorarea) deoarece folosește entități grafice.
Un Mindmap poate fi desenat de mână sau cu ajutorul unei aplicații software.

Utilizări ale Mind map:
- Organizarea structurii de cunoștințe și clasificarea informației; Prezentarea informației – o mai bună înțelegere a unui text și memorarea ideilor principale; Învățare și memorare; Planificare (timp de studiu, evenimente, prezentări, proiect); Rezolvarea de probleme prin îmbunătățirea funcțiilor cognitive de gândire;
- Brainstorming – Creativitate; Luarea de notițe și sintetizarea unui text sau a unei prezentări; Luarea de decizii;

Două hărţi mentale care reprezintă același subiect, dar au fost realizate de două persoane diferite, vor arăta diferit. Acest lucru se întâmplă deoarece fiecare persoană are propriile modalități de gândire logică și propriile cunoștințe adunate din experiență care se vor reflecta în Mindmap. De asemenea, o aceeași persoană poate să realizeze, pentru același subiect, reprezentări diferite la momente de timp diferite. (extras http://rom.explainwell.org/)
Vizionare filme pe tema mindmapping.

How to Mind Map with Tony Buzan -
https://www.youtube.com/watch?v=u5Y4pIsXTV0
Want to learn better? Start mind mapping | Hazel Wagner
https://www.youtube.com/watch?v=5nTuScU70As
Why people believe they can't draw - and how to prove they can
https://www.youtube.com/watch?v=7TXEZ4tP06c

Exerciţiul 1. “Pot”-“Nu pot”
Timp alocat: 10 minute;
Activitate:
În 60 de secunde vă rog să notaţi cât mai multe lucruri pe care le puteţi face cu un obiect (alegeţi obiectul pe care îl doriţi).
În 60 de secunde vă rog să notaţi cât mai multe lucruri pe care nu le puteţi face cu obiectul ales mai sus.



Exerciţiul 2. Funcţiile emisferelor cerebrale
Timp alocat: 10 minute
Organizare: individual

Activitate: Vă rog să notaţi cât mai multe activităţi pe care puteţi să le faceţi şi pe care le puneţi pe seama abilităţilor stimulate de emisferele cerebrale.

Exerciţiul 3. Planetele sistemului solar
Timp alocat: 5 minute
Activitate 1: Scrieţi cât de repede puteţi planetele sistemului solar în ordinea depărtării de soare.
Activitate 1: Exerciţiu de memorare

Exerciţiul 4. Creare hartă mentală
Timp alocat: 5 minute
Activitate: Creaţi o hartă mentală (mind map) pentru tema din partea introductivă: Gândire creativă şi informatică.

Exerciţiul 5: Creare hartă mentală
Creaţi o hartă mentală (mind map) pentru tema Scriere academică şi profesională.
Timp alocat: 10 minute
Materiale: Legile creării hărţilor mentale (Tony Buzan)

Exerciţiul 6: Programe+ aplicaţii software mind mapping
Vă invit să căutaţi un program-aplicaţie software pentru crearea de hărţi mentale şi să exersaţi conceperea unei hărţi pe tema Scriere academică şi profesională sau pe o temă la alegere.

THE LAWS OF MIND MAPPING

(Text original în limba engleză - Tony Buzan)

The Mind Mapping laws are designed to help you more rapidly gain access to your intelligence by giving you specific techniques that are brain-compatible. By following the laws, your memory and creativity will be enormously enhanced.

Law One: A Mind Map commences in the center of a page within a multi-colored image or symbol.

Reasons: It commences in the center because this reflects the many-hooked nature of the brain's thinking processes, and allows more space and freedom for developing ideas from the central core. Use image and color because the old adage, "a picture is worth a thousand words" applies here in both memory and creativity.

Law Two: Main themes are attached to the central image on six lines using large capital letters.

Reasons: Main themes are attached because the brain works by association, and if the lines are attached the ideas will internally be similarly "attached." The lines are thicker and the printing larger to reflect the importance of these ideas.

Law Three: Lines are connected to lines.

Reasons: The connected structure of the Mind Map reflects the associative nature of the brain.

Law Four: Words are printed.

Reasons: Printing the words may take slightly longer in execution, but the immediate "photographic feedback" and comparative clarity of the printed word give enormous advantage.

Law Five: Words are printed on lines.

Reasons: Printing the words on the line gives them connection and association to the basic structure of the Mind Map. People often find that if they can reconstruct the general skeleton of the Mind Map, the words immediately "pop in" to place.

Law Six: Single key words per line.

Reasons: Each key word has its own million-range of possibilities for association. Placing the key word alone on a line gives the brain more freedom to branch out from that word. Phrases trap the individual word, and reduce the possibilities for creativity and the clarity of memory.

Law Seven: Use of color throughout the Mind Map.

Reasons: Color is a major stimulator of all forms of thought, and especially enhances creativity and memory. It also appeals to aesthetic sensitivities which increase the brain's pleasure in building the Mind Map, and its interest in returning to, reviewing and using it

Law Eight: Images throughout the Mind Map.

Reasons: As Leonardo da Vinci recommended for appropriate brain training: "Learn the Science of Art." The use of images can raise memory performance to near perfect, multiplies creative thinking effectiveness by as much as ten times, and improves problem solving and communications, et cetera. It also, over time, increases the individual's perceptual capabilities and skills.

Law Nine: Use of codes and symbols throughout.

Reasons: Personalized codes using various shapes such as colors and arrows add a "fourth dimension" to a Mind Map. They greatly enhance the Mind Mapper's ability to analyze, define, structure, organize and reason.

General Principles:

As Mind Mapping is a process of accessing and using the major cortical skill areas, it is important to have a "brain-supportive" environment.

Wherever possible, the environment should have the following characteristics:

1. **Natural light**
2. **Fresh air**
3. **High quality Mind Mapping material**
4. **Good work space**
5. **Posture-enhancing chairs**
6. **Physical objects and decorations that feed the senses**
7. **Temperature control**

It is also recommended that the Mind Mapper organize at an early stage in the development of his/her Mind Mapping skill a filing, cross-referencing and retrieval system for Mind Maps of differing sizes, and for those which cover different subject areas.

With each Mind Map, it is useful to have as a continuing goal the ongoing development of artistic and organizational skills. This makes the Mind Map not only specifically project-oriented, but also an ongoing tool for personal mental development on all levels.