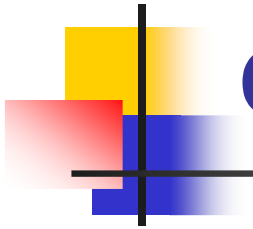
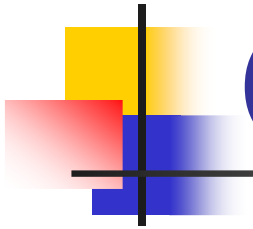
Елементарне методе сортирања



- Сортирање низа у неопадајућем или нерастућем поретку је поступак проналажења једне пермутације елемената низа у којој се елементи појављују у неопадајућем тј. нерастућем поретку
- Елементарне методе сортирања су временске сложености $O(n^2)$
- Једноставне су али неефикасне за велике скупове ($n > 500$)
- Омогућавају изучавање сложенијих
  - Сортирање избором највећег (најмањег) елемента (selection sort)
  - Сортирање уметањем (insertion sort)

Сортирање избором највећег (најмањег) елемента



- Сортирање низа (x_n) од n елемената
- Поступак
  - Највећи елемент низа доводи се на прво место
  - Највећи од преосталих n-1 елемената доводи се на друго место
  - Највећи од преосталих n-2 елемената доводи се на треће место, ...
  - Већи од преостала два елемента доводи се на претпоследње место
  - На последњем месту остаје најмањи елемент

Сортирање избором највећег (најмањег) елемента

- Глобални алгоритам
  - (а) одредити највећи елемент x_{max}
  - (б) разменити x_{max} и x_1
  - (в) поновити кораке (а) и (б) посматрајући низове $x_2,...,x_n$; $x_{3,...,}$ x_n; ... x_{n-1},x_n

- Псеудопрограм

```
for (i=1; i<n; i++)
{
        naći najveći element u nizu x[i], x[i+1],...,x[n] i zapamtiti
njegov indeks max;
        razmeniti x[i] i x[max];
}
```

Налажење максималног елемента низа

- Нека је дат низ $x[m..n]$ Задатак је наћи индекс j т.д. $x_j=\max(x_m,\ldots,x_n)$
- Поступак
  - Први у низу се прогласи за највећи (max = *m)*
  - Највећи се пореди са свим осталим елементима (са индексима од $m + 1$ до n)
  - Сваки елемент већи од највећег проглашава се за највећи а његов индекс *j* за индекс највећег, *max* (*max* = *j*)

Налажење максималног елемента низа: функција

```
int maxelem(int x[ ], int m, int n)
{
   int max;
   max = m;
   for (j=m+1; j <= n; j++)
        if (x[j] > x[max]) max=j;
   return max;
}
```

Функција сортирање избором највећег (најмањег) елемента

```
void swap(int x[ ], int i, int j);
void maxelem(int x[ ], int m, int n);
void maxsort(int x[ ], int n)
{
    int i,j,max;
    for(i=0; i<n–1; i++)
    swap(x, i, maxelem(x, i, n–1));
}
void swap(int x[ ], int i, int j)
{
    int priv;
    priv = x[i];
    x[i] = x[j];
    x[j] = priv;
}
```

Сортирање избором највећег (најмањег) елемента: сложеност

- Добра својства за низове до 1000 елемената
- Број поређења међу елементима низа је око $n^2/2$ јер се спољашња петља извршава n пута а унутрашња око $n/2$ пута у просеку
- Укупно време извршавања алгоритма је пропорционално *са* n^2

Сортирање уметањем: претпоставка, поступак

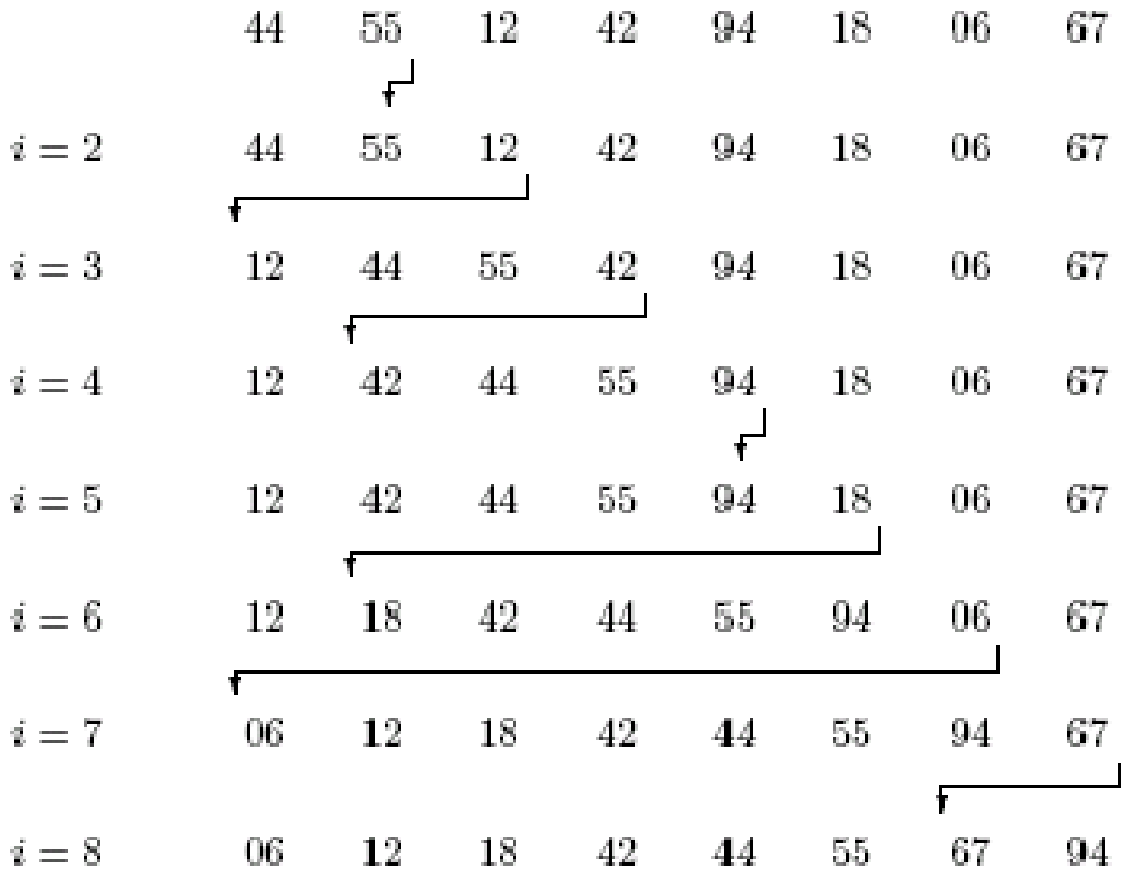
- Имамо уређен почетни део низа $x_1,\ldots,x_{i-1}$
- То увек важи за *i=2*
- У сваком кораку, од *i=2,* i-ти елемент се ставља на право место у односу на првих (уређених) *i-1*

- Глобални алгоритам

```
for (i=2; i<=n; i++) {
    v = x[i];
    ”umetnuti” v na odgovarajuće mesto u x1,x2,…, xi
}
```

  - “Уметање” на одговарајуће место: померањем елемената који претходе ($x[j]$, $j = i-1,\ldots,1$) за по једно место удесно ($x[j+1] = x[j]$)
  - Критеријум краја :
    - Нађена вредност x_j *т.д.* $x_j < v$ (v се ставља на j+1. место)
    - Дошло се до левог краја низа

Сортирање уметањем: пример



Сортирање уметањем: функција

```
void umetsort(int x[ ], int n)
{
    int v;
    for(i=1; i<n; i++) {
        v=x[i]; j=i–1;
        while (j>=0 && v<x[j]) {
            x[j+1]=x[j]; j--;
        }
        x[j+1]=v;
    }
}
```

Сортирање уметањем: сложеност

- Број поређења, P_i у i–том пролазу је највише i а најмање 1, у просеку је $i/2$.
- Број премештања, M_i је P_i+1
- Укупан број поређења и премештања је
- P_{min} = n-1
- P_{max} = n + (n-1) + (n-2) + … + 1 = n*(n+1)/2 = ½*(n^2+n)
- И слично за M_i
- Најмањи број поређења и премештања је кад је полазни низ уређен (O(n))
- Највећи број поређења и премештања је кад је полазни низ уређен у обрнутом поретку (O(n^2))
- Могућа побољшања алгоритма користе уређеност низа $x_1, x_2, \ldots, x_{i-1}$ за бинарну претрагу – сортирање бинарним уметањем

Рекурзивно сортирање уметањем

```
#include <stdio.h>
 #define N 7
 void InsertionSort(int, int [], int );
int main(void)
{
    int i;
    int x[N] = {37,23,17,12,72,31,46};

    InsertionSort(1, x, N);
    printf("Sortirani niz brojeva je\n");
     for (i = 0; i < N; i++)
        printf("%d ",x[i]);
    printf("\n\n");

    return 0;
}
```

Рекурзивно сортирање уметањем

```
void InsertionSort(int i, int x[], int n)
{
    if (i < n)
    {
        int j;
        int v = x[i];

        for (j = i; j > 0 && x[j-1] > v; j--)
            x[j] = x[j-1];
        x[j] = v;

        InsertionSort(++i, x, n);
    }
}
```

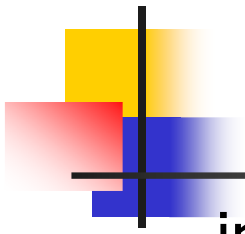
Рекурзивно сортирање избором

```
#include <stdio.h>
#define N 7
 void SelectionSort(int [] , int , int );
int findMax(int x[], int n, int m);
int main(void)
{
    int i;
    int x[N] = {37,23,17,12,72,31,46};

    SelectionSort(x, N, 0);
    printf("The list of sorted numbers is\n");

    for (i = 0; i < N; i++)
        printf("%d ",x[i]);
    printf("\n\n");
    return 0;
 }
```

Рекурзивно сортирање избором



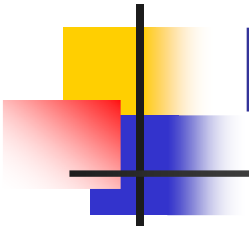
```
int findMax(int x[], int n, int m)
{
  int maxIndex;
  if (m == n - 1) return m;
  maxIndex = findMax(x, n, m + 1);

  if (x[m] > x[maxIndex])
    return m;
  else
    return maxIndex;
}
```

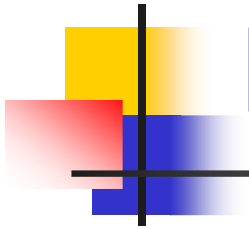
Рекурзивно сортирање избором

```
void SelectionSort(int x[], int n, int m)
{  int maxIndex;
    int temp;
    if ( m >= n - 1 )
        return;
    maxIndex = findMax(x,n,m);
    temp = x[m];
    x[m] = x[maxIndex];
    x[maxIndex] = temp;
    SelectionSort(x, n, m + 1);
}
```

Брзи сорт (Quicksort)



- Најчешће коришћен алгоритам сортирања
- Основни облик алгоритма: британски научник Ричард Хор (Sir Charles Antony Richard Hoare), 1960. године, са 26 година
- Једноставна имплементација
- Захтева мање простора и времена од других алгоритама сортирања, у већини случајева
- Просечно $n\log n$ операција за сортирање низа од n елемената
- Лоше стране:
  - Рекрзиван (нерекурзивна варијанта много сложенија)
  - У најгорем случају извршава око n^2 операција



Брзи сорт (Quicksort): идеја

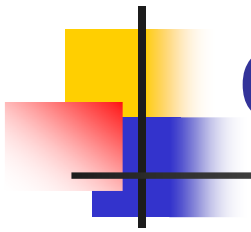
- Партиционисање низа према одабраном елементу партиционисања (“пивот”)
- Елемент партиционисања доводи се на право место у низу
- Исти алгоритам се примењује (рекурзивно) на сваку од две добијене партиције
- Рекурзивни позив се завршава када се примени на партицију са мање од два елемента
- Параметри, поред низа, су и индекси крајњег левог и десног елемента

Брзи сорт (Quicksort): функција

```
void quicksort (int x[ ], int l, int d)
{
    int i;
   if (d > l) {
        i=partition(l,d);
        quicksort(x, l, i–1);
        quicksort(x, i+1, d);
   }
}
```

- Позив функције quicksort(x,1,*n*) сортира цео низ

Брзи сорт (Quicksort): функција partition - својства



- Преуређује низ тако да важи:
  - $x[i]$ је на свом месту за неко i
  - сви елементи $x[l],\ldots,x[i-1]$ су мањи или једнаки са $x[i]$
  - сви елементи $x[i+1],\ldots,x[d]$ су већи или једнаки са $x[i]$

Брзи сорт (Quicksort): функција partition - својства

- Једна могућа стратегија:
  - Произвољно одабрати $x[d]$ као елемент партиционисања (иде не коначну позицију)
  - Пролази се кроз низ слева док се не наиђе на елемент већи од $x[d]$ (позиција *левог маркера*)
  - Пролази се кроз низ здесна док се не наиђе на елемент мањи од $x[d]$ (позиција *десног маркера*)
  - Два елемента која су зауставила пролаз размењујемо
  - Лево од левог маркера су елементи мањи од (или једнаки са) $x[d]$, десно од десног – већи од (или једнаки са) $x[d]$
  - Настављамо поступак док се маркери не сретну
  - Разменити $x[d]$ са елементом на коме су се маркери срели

Брзи сорт (Quicksort): пример партиционисања

- Елемент партиционисања је подвучен

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
P	R	I	M	E	R	Z	A	S	O	R	T	I	R	A	NJ	E
A	R	I	M	E	R	Z	A	S	O	R	T	I	R	P	NJ	E
A	A	I	M	E	R	Z	R	S	O	R	T	I	R	P	NJ	E
A	A	E	M	E	R	Z	R	S	O	R	T	I	R	P	NJ	I

Брзи сорт (Quicksort): сложеност

- Лоше карактеристике – простор и време - када се примењује на већ сортирани низ (око $n^2/2$ операција и n локација за обраду рекурзије
- Најбољи случај: у свакој фази партиционисања низ се дели тачно на пола
- Примена “подели па владај” стратегије:
- $C(n) = 2*C(n/2)+n \quad (C(1) = 1)$
- За процену вредности $C(n)$ бирају се вредности за n облика 2^N

$$C(2^N) = 2 * C(2^{N-1}) + 2^N \qquad /2^N$$
$$\frac{C(2^N)}{2^N} = \frac{C(2^{N-1})}{2^{N-1}} + 1$$

- Примењујући исту формулу N пута добије се N копија “1”, тј.

$$C(2^N) = N * 2^N, \text{ тј. } C(n) \approx n \ln n$$

- Може да се докаже да је и средњи случај $O(n \ln n)$

Брзи сорт (Quicksort): једна имплементација, поступак

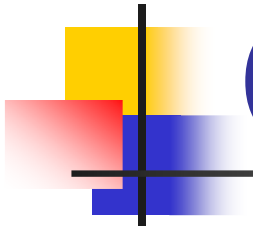
- "The Programming Language C" B. Karnighan, D. Ritchie

- Елемент партиционисања: средишњи елемент
- *На почетку се "склања" на крајњу леву позицију (размењује се са крајњим левим)*
- Пролази се слева на десно и сваки елемент који је мањи од елемента партиционисања премешта се у леви део низа
- *Променљива "последњи" памти највећи индекс елемента мањег од елемента партиционисања*
- Када се прође кроз све елементе низа, елемент партиционисања се размењује са елементом на позицији "последњи" (и поставља на своју коначну позицију)
- *Затим се функција Quicksort позива рекурзивно за леви и десни подниз*

Брзи сорт (Quicksort): једна имплементација, функција

```
void qsort(int v[], int l, int d)
{
    int i, poslednji;
    void swap(int v[], int i, int j);
    if(l>= d) return;
    swap(v,l,(l+d)/2);
    poslednji=l;
    for(i=l+1; i<=d; i++)
            if (v[i]< v[l]) swap(v, ++poslednji, i);
    swap(v, l poslednji);
    qsort(v, l , poslednji-1);
    qsort(v, poslednji+1, d);
}
```

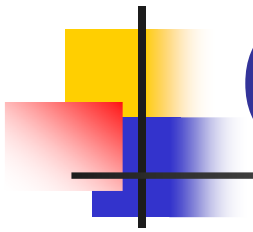
Сортирање спајањем (Merge sort)



- Рекурзивни алгоритам заснован на поређењу
- Још један пример алгоритамске парадигме “подели па владај”
- Џон фон Нојман (John von Neumann) – мађарско-амерички математичар, 1945.
- Временска сложеност пропорционална је са $n\log n$ а (додатна) просторна сложеност са n.
- У већини имплементација је стабилан

- **Поступак**:
  - Поделити несортирани низ у два подниза приближно једнаке дужине, *levi* и *desni*
  - Сортирати сваки подниз рекурзивно истим алгоритмом
  - Спојити два сортирана подниза у један сортирани низ (или дописати елементе подниза *desni* на подниз *levi* ако се тако добије сортирани низ)

Сортирање спајањем (Merge sort): принципи



1. Кратки низ је могуће сортирати брже него дугачки (основа за дељење на два подниза)

2. Брже се гради сортирани низ од сортираних поднизова него од несортираних (основа за спајање)

Сортирање спајањем (Merge sort): псеудопрограм

```
algoritam merge_sort(A)
{
    tip_elementa_niza levi[ ], desni[ ], rezultat[ ];
    int sredina;
    if (length(A) <= 1)
            return A;
    sredina = length(A) / 2;
    for (i=0; i<=sredina; i++)
            upisati A[i] u niz levi;
    for (i=sredina+1; i<= length(A)-1; i++)
            upisati A[i] u niz desni;
    levi = merge_sort(levi);
    desni = merge_sort(desni);
    if levi[sredina] > desni[0]
            rezultat = spoji(levi, desni);
    else
            rezultat = dodaj(levi, desni;)
    return rezultat;
}
```

Сортирање спајањем (Merge sort): функција *spoji*

```
function spoji(levi,desni)
{
	/* niz levi ima length(levi) elemenata: levi[0..length(levi)-1] */
	/* niz desni ima length(desni) elemenata: desni[0..length(desni)-1] */
	/* niz rezultat imaće length(levi)+length(desni) elemenata */
	tip_elementa_niza rezultat[ ];
	while length(levi) > 0 && length(desni) >0
		if levi[0] <= desni[0]
		{
			upiši levi[0] u niz rezultat;
	/* iz niza levi isključuje se prvi element - onaj sa indeksom 0 */
			levi = levi[1..length(levi)-1];
		}
		else
		{
			upiši desni[0] u niz rezultat;
	/* iz niza desni isključuje se prvi element - onaj sa indeksom 0 */
			desni = desni[1..length(desni)-1];
		}
	if length(levi) > 0
		upiši ostatak niza levi u niz rezultat;
	else
		upiši ostatak niza desni u niz rezultat;
	return rezultat;
}
```

Сортирање спајањем (Merge sort): пример