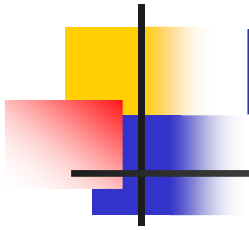
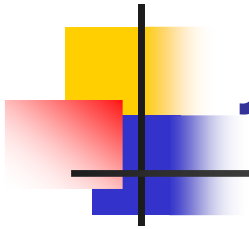
# Претраживање и сортирање (Програмирање 2, глава 5, тачке 5.2, 5.3)



# Претраживање

- Претраживање датог низа елемената:
- одређивање индекса елемента низа који је једнак датој вредности или испуњава неко друго својство (нпр. највећи је елемент низа, први непарни елемент низа, први број већи од задате вредности и сл.)



# Линеарно претраживање

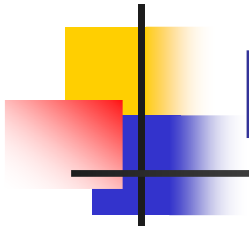
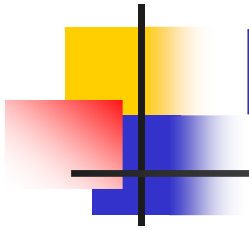
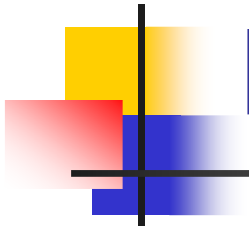
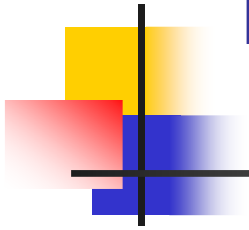
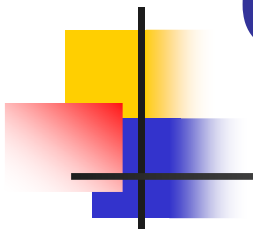
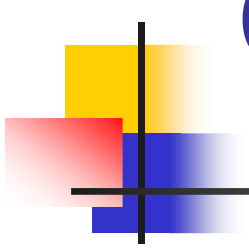
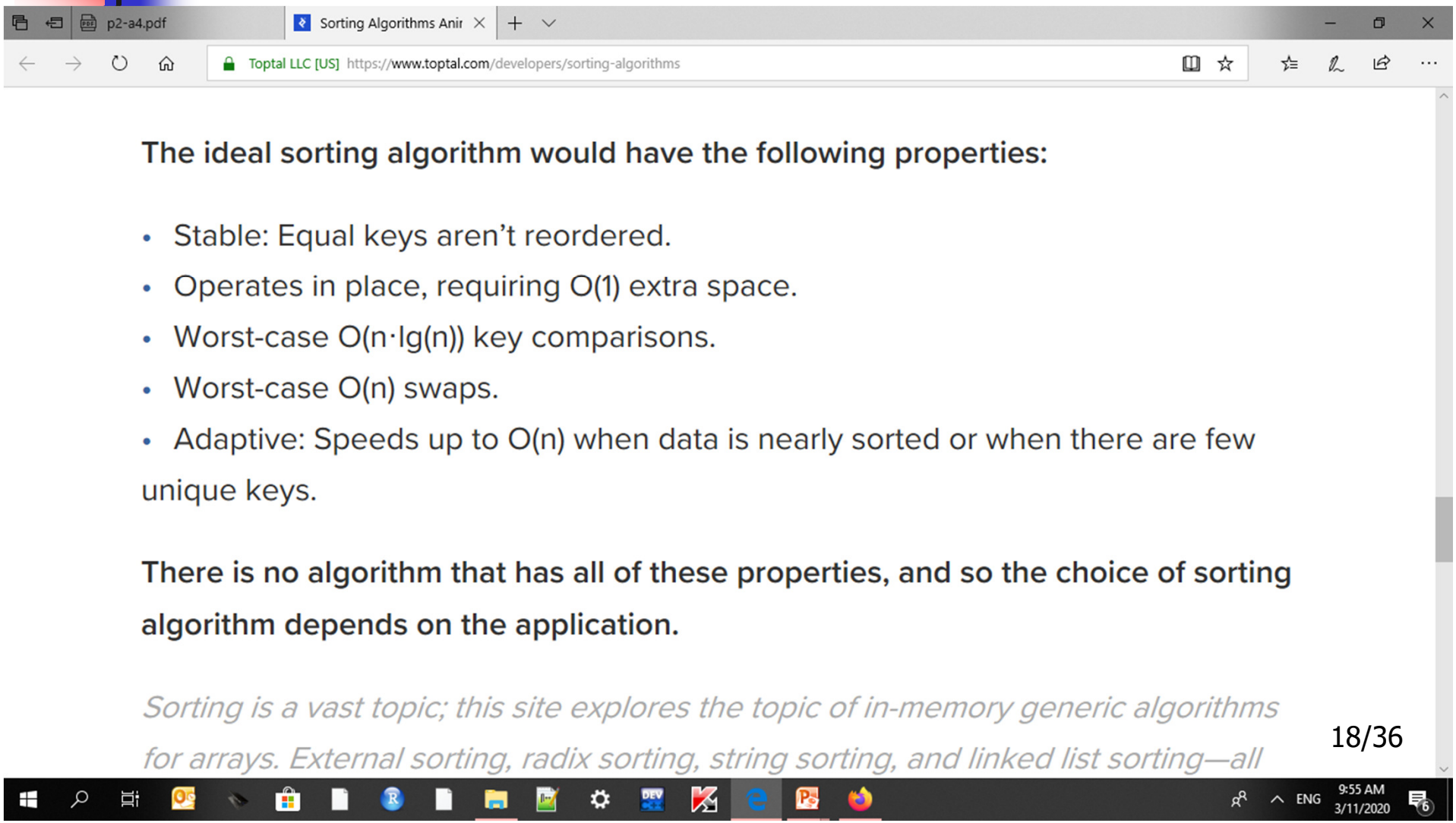
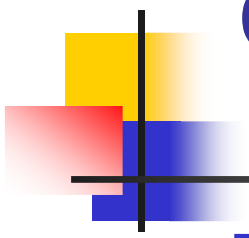
- Линеарно (или секвенцијално) претраживање низа састоји се у испитивању редом свих елемената низа (почевши од елемента са најмањим индексом), или док се не наиђе на тражени елемент (задату вредност или елемент који има неко тражено својство)

- Линеарне временске сложености по броју елемената низа који се претражује, $n$

- У просечном случају испитује се $n/2$ елемената, у најбољем 1, а у најгорем $n$ елемената низа

# Линеарно претраживање – итеративна варијанта

- Итеративна варијанта линеарног претраживања *првог појављивања* задатог целог броја x у задатом низу a дужине *n*:

- ```
int linearna_pretraga(int a[], int n, int x) {
        int i;
        for (i = 0; i < n; i++)
                if (a[i] == x)
                        return i;
        return -1;
}
```
```

# Линеарно претраживање – итеративна варијанта

- Честа грешка при имплементацији линеарног претраживања је прерано враћање негативне вредности:
- int linearna_pretraga(int a[], int n, int x) {

```
        int i;
        for (i = 0; i < n; i++)
                if (a[i] == x) return i;
                else return -1;
}
```

# Линеарно претраживање – рекурзивна варијанта

- *Рекурзивна варијанта* линеарног претраживања ***првог*** *појављивања* задатог целог броја x у задатом низу a дужине *n*
- Параметар *i* који памти позицију од које почиње претраживање:
- int linearna_pretraga(int a[], int i, int n, int x) {

```
    if (i == n) return -1;
    if (a[i] == x) return i;
    return linearna_pretraga(a, i+1, n, x);
}
```

- Репна рекурзија, може једноставно да се трансформише у итеративну варијанту

# Линеарно претраживање – рекурзивна варијанта



- ... линеарног претраживања ***последњег*** *појављивања* задатог целог броја x у задатом низу a дужине *n*
- Једноставно и без параметра
- int linearna_pretraga(int a[], int n, int x) {

```
    if (n == 0) return -1;
    else if (a[n - 1] == x) return n-1;
    else return linearna_pretraga(a, n-1, x);
}
```

- Репна рекурзија, једноставно се трансформише у итерацију



# Бинарно претраживање

- Бинарно претраживање је претраживање низа које претпоставља ***да су елементи низа сортирани***
- У супротном бинарно претраживање није могуће
- У сваком кораку, све док се не нађе тражени елемент, низ се дели на два дела и претраживање се наставља само у једном делу – одбацује се део који сигурно не садржи тражену вредност
- Пример: погађање броја са познатом / непознатом горњом границом
- Сложеност:  O(log n)



# Бинарно претраживање

- Сортираност речника, телефонских именика, енциклопедија
- Варијанта: интерполирано претраживање

# Бинарно претраживање – итеративно решење

```
int binarna_pretraga(int a[], int n, int x) {
        int l, d, s;
        l = 0; d = n-1;
        while(l <= d) {
                s = l + (d - l)/2;
                if (x == a[s]) return s;
                if (x > a[s]) l = s + 1;
                else /* if (x < a[s]) */
                        d = s - 1;
        }
        return -1;
}
```

# Бинарно претраживање – итеративно решење

- Провера једнакости на крају (зашто?):
- int binarna_pretraga(int a[], int n, int x) {

```
        int l, d, s;
        l = 0; d = n-1;
        while(l <= d) {
                s = l + (d - l)/2;
                if (x > a[s]) l = s + 1;
                else if (x < a[s]) d = s - 1;
                else /* if (x == a[s]) */
                        return s;
        }
return -1;
}
```

# Бинарно претраживање – рекурзивно решење

- Увођење параметара леве и десне границе (репна рекурзија) :
- int binarna_pretraga(int a[], int l, int d, int x) {

```
        int s;
        if (l > d) return -1;
        s = l + (d - l)/2;
        if (x == a[s]) return s;
        if (x < a[s]) return binarna_pretraga(a, l, s-1, x);
        else /*if (x > a[s])*/
                return binarna_pretraga(a, s+1, d, x);
}
```

# Интерполирано претраживање

- Модификација бинарног:
- Уместо s = l+ (d-l)/2 = l+(1/2)*(d-l)),
- може да се користи, на пример,

- s=l+(x-b[l])*(d-l) / (b[d] - b[l])
- Пример:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | R | I | M | E | R | P | R | E | T | R | A | Ž | I | V | A | NJ | A |

- Сортирани низ:

| A | A | A | E | E | I | I | M | NJ | P | P | R | R | R | R | T | V | Ž |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# Интерполирано претраживање: пример

- Нека се тражи елемент са вредношћу V:
- Бинарно претраживање:
- A A A E E I I M NJ P P R R R R T V
- P P R R R R T V Z
- R T V Z
- V Z
- Кодирање бројевима!!!
- Интерполирано претраживање:

| ind. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| el. | | A | A | A | E | E | I | I | M | NJ | P | P | R | R | R | R | T | V | Ž |
| kod: | 1 | 1 | 1 | 9 | 9 | 13 | 13 | 18 | 20 | 22 | 22 | 23 | 23 | 23 | 23 | 26 | 28 | 30 |
| | | | | | | | | | | | | | | | | | V | Ž |

# Интерполирано претраживање: сложеност



- loglog $n$

# Сортирање



- Уређивање низа у односу на неко линеарно уређење

- Примери:
- Уређење низа бројева по величини растуће или опадајуће,
- Уређивање низа ниски лексикографски или по дужини,
- Уређивање низа структура на основу вредности неког поља

# Сортирање - алгоритми



- Bubble sort
- Selection sort
- Insertion sort
- Shell sort
- Merge sort
- Quick sort
- Heap sort
- ... разлике...
- Временске / просторне сложености
- http://www.sorting-algorithms.com/

# Сортирање - алгоритми



# Сортирање – класе сложености



- Једноставнији, спорији алгоритми имају сложеност најгорег случаја $O(n^2)$:
  - bubble
  - insertion
  - selection
- Shell sort, зависно од имплементације, $O(n^2)$ до $O(n \log^2 n)$
- Бржи алгоритми имају сложеност најгорег случаја $O(n \log n)$ (додатни меморијски простор)
  - heap sort
  - merge sort
- quick sort има сложеност најгорег случаја $O(n^2)$, али просечног $O(n \log n)$; убраја се у веома брзе алгоритме

# Selection sort

- Идеја: пронаћи најмањи елемент у низу и поставити га на нулто место
- Затим пронаћи најмањи елемент у остатку низа и довести га на прво место,
- Тако редом до краја низа
- Индекс најмањег елемента у низу почевши од позиције *i* :
- int poz_min(int a[], int n, int i) {

```
        int m = i, j;
        for (j = i + 1; j < n; j++)
                if (a[j] < a[m])
                        m = j;
        return m;
}
```

# Selection sort

- void razmeni(int a[], int i, int j) {
  int tmp = a[i]; a[i] = a[j]; a[j] = tmp;
  }
- int selectionsort(int a[], int n) {
  int i;
  for (i = 0; i < n - 1; i++)
  razmeni(a, i, poz_min(a, n, i));
  }
- Временска сложеност је квадратна (броје се само поређења и размене), просторна је константна
- („сортирање у месту“)

# Selection sort – рекурзивна варијанта

- Нешто једноставнија имплементација ако се највећи елемент доводи на крај низа

- ```
  int poz_max(int a[], int n) {
          if (n == 1)
                  return 0;
          else {
                  int m = poz_max(a, n-1);
                  return a[m] > a[n-1] ? m : n-1;
          }
  }
  ```
  ```

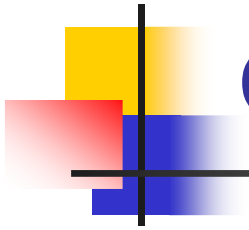
# Selection sort – рекурзивна варијанта

```
void selectionsort(int a[], int n) {
        if (n > 1) {
                razmeni(a, n-1, poz_max(a, n));
                selectionsort(a, n-1);
        }
}
```

# Selection sort - анимација

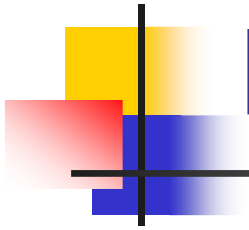
- https://www.toptal.com/developers/sorting-algorithms/selection-sort

# Merge sort - сортирање спајањем



- Рекурзивни алгоритам
- Пример алгоритамске парадигме “подели па владај”
- Џон фон Нојман (John von Neumann) – мађарско-амерички математичар, 1945.
- Временска сложеност је **O(*n*log*n*)** у најгорем случају а просторна сложеност са ***O(n)***.

- **Поступак**:
  - *Поделити* несортирани низ у два подниза приближно једнаке дужине, *levi* и *desni* (*дужине се разликују највише за 1*)
  - *Сортирати* сваки подниз *рекурзивно* истим алгоритмом
  - Излаз из рекурзије је јединични низ
  - *Спојити* два сортирана подниза у један сортирани низ (или дописати елементе подниза *desni* на подниз *levi* ако се тако добије сортирани низ) – временска сложеност овог корака је *O(n)*



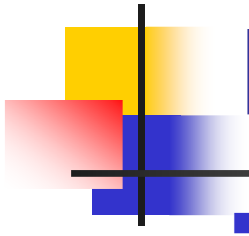
# Merge sort: принципи

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)

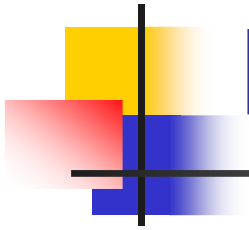
# Merge sort: спајање сортираних низова

```
void merge(int a[], int m, int b[], int n, int c[]) {
        int i, j, k;
        i = 0, j = 0, k = 0;
        while (i < m && j < n)
                c[k++] = a[i] < b[j] ? a[i++] : b[j++];
        while(i < m) c[k++] = a[i++];
        while(j < n) c[k++] = b[j++];
}
```



# Merge sort

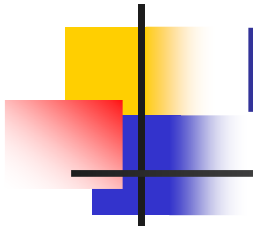
```
void mergesort_(int a[], int l, int d, int tmp[]) {
        if (l < d) {
                int i, j;
                int n = d - l + 1, s = l + n/2;
                int n1 = n/2, n2 = n - n/2;
                mergesort_(a, l, s-1, tmp);
                mergesort_(a, s, d, tmp);
                merge(a + l, n1, a + s, n2, tmp);
                for (i = l, j = 0; i <= d; i++, j++)
                        a[i] = tmp[j];
        }
}
```



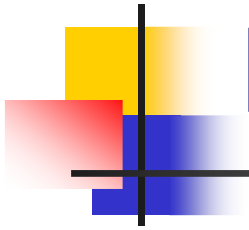
# Merge sort: анимација

- https://www.toptal.com/developers/sorting-algorithms/merge-sort

# 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
- анимација: https://www.toptal.com/developers/sorting-algorithms/quick-sort

# Брзи сорт (Quicksort): функција

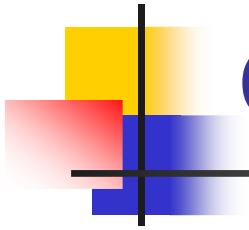
```
void quicksort (int x[ ], int l, int d)
{

   if (l<d) {
         int p =partition(a,l,d);
         quicksort(x, l, p–1);
         quicksort(x, p+1, d);
   }
}
```

- Позив функције quicksort(x,0,*n-1*) сортира цео низ

# Брзи сорт (Quicksort): функција partition - својства

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

# Брзи сорт (Quicksort): функција partition - идеја



- *Пивот (елемент партиционисања) се “склања” на крајњу леву позицију (размењује се са крајњим левим) или је баш крајњи леви*
- Пролази се слева на десно и сваки елемент који је мањи од елемента партиционисања премешта се у леви део низа
- *Променљива “p” памти највећи индекс елемента мањег од елемента партиционисања*
- Када се прође кроз све елементе низа, елемент партиционисања се размењује са елементом на позицији “p” (и поставља на своју коначну позицију)

# Функција partition – једна имплементација

```
int partition (int a[], int l, int d) {
   /*dovedi pivot na krajnju levu poziciju l*/
   swap(a, l, izbor_pivota(a, l, d));
   int p = l, j;
   for (j = l+1; j <= d; j++)
      if (a[j] < a[l]) /*if(a[j] < pivot)*/
         swap(a, ++p, j); /*element koji je manji od pivota
           dovodimo na poziciju poslednjeg takvog*/
   swap(a, l, p); /*dovodimo pivot na njegovu poziciju*/
   return p; /*vracamo poziciju pivota*/s
}
```

# Quicksort - дискусија

- 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:
- $T(n) = 2*T(n/2)+n \qquad (T(1) = 1)$
- tj. O($nlogn$)
- Partition јесте O($n$)
- Избор пивота не може да обезбеди половљење низа
- Зато је најгори случај O($n^2$)
- У просечном случају може да се очекује релативно равномерна расподела што доводи до оптималне временске сложености (O($n \log n$)) и просторне O($\log n$).