

# Алгоритми за сортиране

Лекция 2 по СДА, Софтуерно Инженерство  
Зимен семестър 2021-2022г

# План на днешната лекция

- Организационни въпроси.
  - Входен тест
  - Задание за самостоятелна работа
- Бавни алгоритми за сортиране
- Бързи алгоритми за сортиране

# Резултати от входяща анкета

Времето на провеждане на лекциите удобно ли ви е и ще можете ли да ги посещавате?

Отговор	Средно	Общо
Да	<div><div></div></div> 86%	104
Не	<div><div></div></div> 14%	17
Total responses to question		
	<div><div></div></div> 100%	121/121

Работите ли?

Отговор	Средно	Общо
Да	<div><div></div></div> 17%	20
Не	<div><div></div></div> 83%	101
Total responses to question		
	<div><div></div></div> 100%	121/121

Как предпочитате да гледате лекциите?

Отговор	Средно	Общо
В зала	<div><div></div></div> 26%	32
Дистанционно	<div><div></div></div> 72%	89

Как предпочитате да участвате в упражненията

Отговор	Средно	Общо
На живо	<div><div></div></div> 40%	49
Дистанционно	<div><div></div></div> 58%	72
Total responses to question	<div><div></div></div> 100%	124/121

Разполагате ли с лаптоп, който да може да носите за присъствени контролни/изпит по структури от данни и алгоритми?

Отговор	Средно	Общо
Да	<div><div></div></div> 98%	119
Не	<div><div></div></div> 2%	2
Total responses to question	<div><div></div></div> 100%	121/121

Въз основа на описанието на курса, чувствате ли се подготвени за него?  
Задължително изискване е да владеете материала от "Увод в програмирането"

Отговор	Средно	Общо
Да	<div><div></div></div> 93%	113
Не	<div><div></div></div> 7%	8

Известен ли ви е сайта <https://www.hackerrank.com/> и решавали ли сте задачи в него?

Отговор	Средно	Общо
Да	<div><div></div></div> 45%	54
Не	<div><div></div></div> 55%	67
Total responses to question	<div><div></div></div> 100%	121/121

Занимавали ли сте се със състезателно програмиране?

Отговор	Средно	Общо
Да	<div><div></div></div> 8%	10
Не	<div><div></div></div> 92%	111
Total responses to question	<div><div></div></div> 100%	121/121

Мислите ли, че представеният курс ще ви е полезен за професионалната ви реализация?

Отговор	Средно	Общо
Да	<div><div></div></div> 99%	120
Не	<div><div></div></div> 1%	1

Ще можете ли да отделяте в допълнение към 3-те часа лекции и 2-та часа упражнения още 4-6 часа на седмица за самостоятелна подготовка въкщи?

Отговор	Средно	Общо
Да	<div><div></div></div> 92%	111
Не	<div><div></div></div> 8%	10

Интересувате ли се от летен стаж в голяма технологична компания?

Отговор	Средно	Общо
Да	<div><div></div></div> 86%	104
Не	<div><div></div></div> 14%	17

Самостоятелното писане на програмен код е от изключителна важност за развитие на уменията за програмиране и решаване на алгоритмични проблеми.

С участието си в този курс се съгласявате да не участвате в схеми за преписване или измама. С подобни действия в дългосрочен план ще ощетите хората, на които евентуално краткосрочно "помагате". Ако искате да помогнете на някой обяснявайте му извън лекции, как се решават задачите, но не му ги решавайте. Потвърждавате ли своето съгласие?

Отговор	Средно	Общо
Да	<div><div></div></div> 100%	121
Не	<div><div></div></div>	0

# Задание за самостоятелна работа

120+ студента решили поне една задача

100+ студента са решили всички задачи!

Браво! Продължавайте с добрата работа!

Адрес на следващото домашно:

<https://www.hackerrank.com/contests/sda-hw-2>

# Алгоритми за сортиране



# Какво ще научим?

1. Какво е сортиране и за какво ни е необходимо?
2. “Бавни” алгоритми за сортиране (bubble, selection, insertion)
3. “Бързи” алгоритми за сортиране (merge, quick)
4. Специфични алгоритми за сортиране(count sort)

Какво е сортиране и за какво ни е необходимо?

- Основен клас алгоритми често необходим за решаване на реални проблеми
- Добър пример за демонстриране на това какво е алгоритъм, как се изчислява сложност на алгоритъм, как да бъдем по-критични към това с каква сложност решаваме проблем.

# Дефиниция на сортиране

Проблем: Да се напише процедура, която подрежда в определен ред обекти подадени като вход.

Вход: масив с числа

Изход: масив със същите числа наредени в нарастващ(или намаляващ) ред.

# Сортиране с метод на мехурчето

Основна идея:

Започвайки последователно от началният елемент до крайният сравняваме всеки елемент със следващия като ги разменяме ако не са подредени. По този начин на всяка стъпка изкарваме най-големият в края на правилното за него място. Повтаряйки този процедура толкова пъти колкото е големината на масива постигаме правилно подреждане на целият масив.

# Визуален пример

<https://visualgo.net/en/sorting>

# Сортиране с метод на мехурчето

```
void bubbleSort(int arr[], int n){  
    int i, j;  
    for (i = 0; i < n-1; i++) {
```

```
        // Last i elements are already in place
```

```
        for (j = 0; j < n-i-1; j++)
```

```
            if (arr[j] > arr[j+1])
```

```
                swap(&arr[j], &arr[j+1]);
```

```
        }
```

```
    }
```

```
void swap(int *a, int *b) {  
    int tmp = *a;  
    *a = *b;  
    *b = tmp;  
}
```

# Сортиране с пряка селекция

Намаля броя на разменянията в сравнение с метода на мехурчето!

Основна идея: Търсим най-големият елемент в масива и директно го поставяме на последно място. След това следващият по големина и отново го поставяме на място и т.н. Докато всички се подредят.

# Визуален пример

<https://visualgo.net/en/sorting>



# Сортиране с пряка селекция

```
for (j = 0; j < n-1; j++){  
    int iMin = j;  
    for (i = j+1; i < n; i++){  
        if (a[i] < a[iMin]){  
            iMin = i;  
        }  
    }  
  
    if (iMin != j){  
        swap(a[j], a[iMin]);  
    }  
}
```

# Сортиране с вмъкване

Основна идея: сортиране постепенно на все по-голяма част от масива, като обхождайки несортираната част всеки един елемент го поставяме в сортираната част с намиране на правилното за него място на което сортираният масив остава сортиран.

# Визуален пример

<https://visualgo.net/en/sorting>

# Сортиране с вмъкване

```
void insertionSort(int arr[], int n) {  
    for (int i = 1; i < n; i++) {  
        int key = arr[i];  
        j = i-1;  
        while (int j >= 0 && arr[j] > key) {  
            arr[j+1] = arr[j];  
            j = j-1;  
        }  
        arr[j+1] = key;  
    }  
}
```

# Сложност на сортиращите алгоритми

Бавни сортировки:

Bubble, Selection, Insertion -  $O(n^2)$

Бързи сортировки:

Merge, Quick -  $O(n \cdot \log(n))$

# Сортиране чрез сливане(merge sort)

Основна идея: Ако имаме два сортирани масива то със линейна сложност може да ги влеем в един масив. Тогава ако разделим масива който искаме да сортираме на по-малки масиви и на всяка стъпка сливаме два по-малки масива в един голям то за  $\log(N)$  стъпки ще слеем всички масиви до един масив, като всяка от стъпките е била линейна.

# Визуален пример

<https://visualgo.net/en/sorting>

# Сортиране със сливане реализация

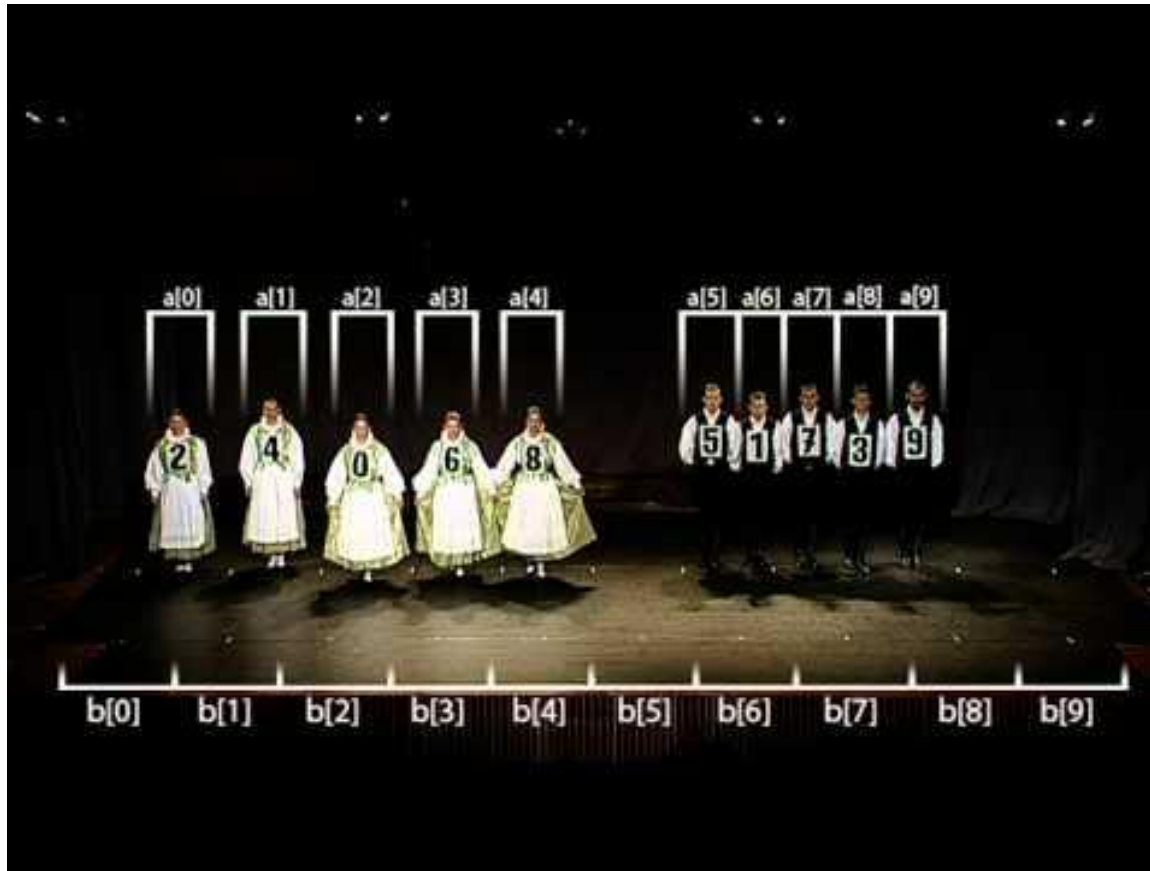
```
void mergesort(int arr[], int l, int r)
{
    if (l < r) //гранично условие на рекурсията
    {
        m= (l+r)/2;
        mergesort(arr, l, m);
        mergesort(arr , m+1, r);
        merge(arr, l, m, r); //функция която слива два масива
    }
}
```



# Сортиране със сливане реализация(2)

```
void merge(int arr[], int start, int middle, int end){  
    // Създаваме масивите arr1 и arr2, които съдържат частите, които ще  
    копитаме. По този начин си освобождаваме основният масив за презаписване.  
  
    i = 0, j = 0, k = start;  
    while( i < middle; j < end-middle){  
        // по малкото число от arr[i] и arr2[j] го записваме в arr[k]  
        // увеличаваме брояча на масива от който копирахме  
        k++; // увеличаваме брояча за основният масив  
    }  
    // допълваме с всички останали необходими елементи от arr1 и arr2  
}
```

# Сортиране със сливане(Забавно видео)



# Merge sort complexity

$O(n \log(n))$  - изчислителна сложност в най-лошият случай

$O(n)$  - сложност по памет

# Бързо сортиране ( quick sort)

Основна идея: Ако вземем едно произволно число от масива, то с линейна сложност можем да прехвърлим всички по-малки числа от масива да са в ляво на числото, а всички по-големи в дясно.

При бързото сортиране избираме число от масива прехвърляме по-малите отляво, по-големите от дясно и след това изпълняваме същата процедура за лявата и дясната половина.

По тази процедура получаваме сложност в средният случай  $O(n \cdot \log(n))$

# Визуален пример

<https://visualgo.net/en/sorting>

# Бързо сортиране реализация

```
void sort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);

        sort(arr, low, pi-1);
        sort(arr, pi+1, high);
    }
}
```

## Бързо сортиране реализация(2)

```
int partition(int arr[], int low, int high){  
    int pivot = arr[high];  
    int i = low; // index of smaller element  
    for (int j=low; j<high; j++){  
        if (arr[j] <= pivot){  
            swap(arr, i, j)  
            i++;  
        }  
    }  
  
    swap(arr, i, high)  
    return i;  
}
```

1  
1  
N \* 5

# Сложност

Сложност в средният случай  $O(n \cdot \log(n))$  ,но....

Сложност в най-лошият случай  $O(N^2)$  - когато масива е сортиран наобратно!



# Рандомизирано Бързо Сортиране

Справя се с проблема, че точно определена редица прави сложността  $O(n^2)$ , като използва произволно избиране на елемент за разделяне.

```
int random_partition(int arr[], int low, int high){  
    swap(arr, high, random(arr.length));  
    partition(arr, low, high);  
}
```

# Merge sort or quick sort

Merge sort изисква  $O(n)$  допълнителна памет, докато quicksort не изисква допълнителна памет.

Merge sort е със сложност в най-лошият случай  $O(n \log(n))$ , Quicksort -  $O(n^2)$

Quicksort - по-бърз от mergesort за малки масиви, но mergesort е по-добър за големи масиви.

# Сортиране с броене (Counting sort)

Можем ли да сортираме със сложност по-малка от  $O(n \cdot \log(n))$ ?

Отговор: Да, но с добавяне на допълнителни ограничения.

При сортирането с броене сложността се определя от броя на различните елементи, които може да има в масива.

# Сортиране с броене

Основна идея: Понеже имаме ограничен брой различни стойности в масива то може да преброим по колко пъти се среща всяка една от тези стойности(с едно обхождане на масива) и след това със второ обхождане да наредим стойностите по техният ред.

Стабилност на сортирането - ако имаме два елемента които са равни в първоначалният масив, то във финалния те се срещат във същият ред като първоначалният масив.

# Визуален пример

<https://visualgo.net/en/sorting>

```
void counting_sort(char arr[],n) {
    char arr_copy[] = new char[arr.length];
    for (int i = 0; i<arr.length; ++i) {
        arr_copy[i] = arr[i];
    }
    int count[] = new int[256];
    for (int i=0; i<n; ++i) {
        count[arr[i]] = count[arr[i]]+1;
    }
    for (int i=1; i<=255; ++i) {
        count[i] += count[i-1];
    }
    // To make it stable we are operating in reverse order.
    for (int i = n-1; i>=0; i--) {
        arr[count[arr_copy[i]]-1] = arr_copy[i];
        count[arr_copy[i]] = count[arr_copy[i]] - 1;
    }
}
```

# Сложност на сортиране с броене

$O(n+k)$

# Radix Sort (Допълнителен материал)

Основна идея - да използваме подход подобен на сортиране с броене, но да може да го прехвърлим и за големи числа.

При radix sort вместо да броим цели числа ще броим само цифри, като ще сортираме масива подред за всички позиции на цифри(единици, десетици, стотици, хиляди, десетохиляди, и т.н). Понеже сортирането с броене запазва подредбата веднъж сортирани числата по последна цифра, те си остават сортирани и при последващо сортиране по десетици и т.н. до последното сортиране.

<https://visualgo.net/en/sorting>



# Обобщение и следващи стъпки

- Разгледахме основните алгоритми сортиране
- От другият път ще разгледаме алгоритми за търсене
- Другият път(19.10.2021 от 18ч) ще се проведе контролно 1 върху темите сортиране и търсене