

Технически Университет - София

КУРСОВА РАБОТА

по Синтез и анализ на алгоритми

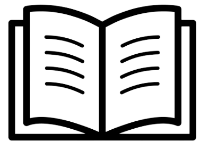
Тема: Игра на пермутация

Изработил: Шериф Бюлентов Пашов

Специалност: Телекомуникации

Група: 57Б

Факултетен №: 111222091

 Съдържание:

1. [Какво е алгоритъм?](#КаквоЕАлогаритъм)

* [Произход на думата „Алгоритъм“](#ПроизходНаДуматаАлгоритъм) …………………………………………………………3
* [Значение на думата „Алгоритъм“](#ЗначениеНаДуматаАлгоритъм) ..……………………………………………………….3
* [Основни етапи при разработката на алгоритми за решаване на практически задачи .](#ОсновниЕтапиПриРазработкатаНаАлгоритми)..................................................................................3
* [Свойства на (компютърните) алгоритми](#СвойстваНаКомпютърнитеАлгоритми) ..................................................3
* [Видове алгоритми](#ВидовеАлгоритми) ......................................................................................5
* [Предимства на алгоритмите](#ПредимстваНаАлгоритмите) .....................................................................5
* [Недостатъци на алгоритмите](#НедостатъциНаАлгоритмите) ....................................................................5

2. [Видове алгоритми за сортиране](#ВидовеАлгоритмиЗаСортиране)

* [Bubble Sort](#BubbleSort) ..................................................................................................6
* [Selection Sort](#SelectionSort) ...............................................................................................6
* [Insertion Sort](#InsertionSort) ...............................................................................................7
* [Counting Sort](#CountingSort) ...............................................................................................7
* [Radix Sort](#RadixSort) .....................................................................................................7
* [Merge Sort](#MergeSort) ...................................................................................................8

3. [Игра на пермутации](#ПравилаНаИгратаНаПермутация)

* [Правила на играта](#ПравилаНаИгратаНаПермутация) ......................................................................................9
* [Алгоритъм написан на Python](#АлгоритъмНаписанНаPython) .................................................................10
* [Примерни вход и изход](#ПримерниВходноИзходниДанни) ...........................................................................11

1. Какво е алгоритъм?

* Произход на думата „Алгоритъм“:   
  Според някои автори думата „алгоритъм“ произлиза от името на персийския учен Ал Хорезми, който през през 825 г. е написал научен трактат за това как да се представят (записват) числата в Десетична бройна система и как се извършват аритметичните операции с тези представяния.
* Значение на думата „Алгоритъм“ :   
  „Алгоритъм“ е интуитивно понятие. То е основно понятие в науката и не се дефинира (както точка, права и равнина в геометрията, пространство и време във физиката и др.), но за него може да се даде интуитивна представа. Понятието „алгоритъм“ е широкообхватно. Разглеждаме алгоритмите, предназначени за изпълнение от компютър. Алгоритъм, това е последователност от стъпки (инструкции, команди, указания, оператори) за действие, която при изпълнението си реализира зададена функционална зависимост между данните и резултатите (иначе казано след изпълнението на предписанието ще бъде решена съответната задача). Ако последователността не реализира зададената функционална зависимост, то тя не е алгоритъм на дадената задача.

Във всяка инструкция (стъпка от алгоритъма) се указват всички или някои от следните елементи:

* Основни етапи при разработката на алгоритми за решаване на практически задачи.

1. Определяне на стратегията на алгоритьма, т.нар. "идеен проект".
2. Формиране и описание на основните относително самостоятелни части, които могат да се

обособят в задачата и алгоритьма за решението й.

1. Определяне на връзките между обектите и процесите в модулите.
2. Разработване на пълен блоков алгоритьм за решаване на задачата с отчитане на необходимите

входни данни и очакваните изходни резултати.

1. Тестова проверка и доказателство на верността на алгоритьма с характеристични набори от

работоспособни данни.

* Свойства на (компютърните) алгоритми:

a) **Детерминираност** (определеност). Алгоритъмът като цяло и всяка негова стъпка при

едни и същи данни дават един и същ (точно определен) резултат при различни

изпълнения. Това свойство се спазва при класическото понятие за алгоритъм, но при

т.нар вероятностни алгоритми то се нарушава. Друго тълкувание за определеност на

алгоритми: всяка стъпка и алгоритъмът като цяло трябва еднозначно да се разбират от

изпълнителя.

б) **Крайност** (финитност). Алгоритъмът и всяка негова стъпка се изпълняват за крайно

време. Алгоритъмът съдържа краен брой стъпки и следователно се изпълнява за краен

брой стъпки.

В математиката има безкрайни процеси. Например, изчисляването на функция чрез

разлагането й в ред на Тейлор тъй като е безкраен процес, то не е алгоритъм. Но ако

функцията се изчислява чрез определен брой начални членове на реда, то безкрайният

процес се превръща в краен и това вече е алгоритъм.

Често поради грешки в алгоритъма (или програмата) процесът се превръща в безкраен,

например при циклите ако е сгрешено условието за излизане от цикъла.

За да се докаже, че дадена последователност от инструкции е алгоритъм, винаги като

доказателство се използва, че тя се изпълнява за крайно време.

в) **Дискретност**. Това свойство е свързано с обстоятелството, че описанието

представено от алгоритъма се състои от краен брой елементи (декларации, обекти,

инструкции и др.), а съответният алгоритмичен процес протича на отделни стъпки.

Изпълнението на алгоритъма във времето се извършва на интервали, на стъпки и всяка

негова стъпка се изпълняват за крайно време. Алгоритъмът съдържа краен брой стъпки

и следователно се изпълнява за краен брой стъпки.

Свойството дискретност налага непрекъснатите по своята природа процеси и обекти да

се моделират чрез дискретни компютърни представяния. А това води до

необходимостта от допълнителни проверки във всеки конкретен случай – доколко

алгоритмичният модел е съответен на (адекватен) на реалния обект/процес.

г) **Масовост**. Това свойство отразява възможността при изпълнението на алгоритъма за

всеки начален елемент (от допустимото множество входни данни) да се получава

търсеният резултат. С други думи: алгоритъмът да може да се прилага не само при

решаването на една конкретна задача, а на цял клас от еднотипни задачи.

д) **Резултатност**. Това свойство означава, че завършването изпълнението на един

алгоритъм е осигурено (за произволни начални данни) след краен брой елементарни

операции. Празно множество резултати е недопустимо!

Резултатността на алгоритъма се третира като насоченост на алгоритъма – след краен

брой стъпки трябва да се получи или решението на поставената задача или отговор за

неприложимостта на този алгоритъм към конкретно избраното множество от началните

данни (което също е резултат). В общия случай е възможно изпълнението на

определения от алгоритъма процес да не завършва (нарушена е крайността) или да

прекъсва на някоя стъпка с резултат „няма решение”.

е) **Цикличност**. Това свойство е присъщо за повечето съвременни алгоритми.

Алгоритъмът съдържа определена последователност от стъпки, която се повтаря

определен брой пъти.

ж) **Формалност**. Не е необходимо изпълнителят да има представа за решаваната задача

и естеството на получаваните резултати – достатъчно е той да изпълнява една след

друга предписаните му елементарни операции (команди). Свойството формалност е от

съществено значение, защото позволява изпълнителят на един алгоритъм да бъде и

автомат.

з) **Изпълнимост**. „Силно” изискване, което се поставя пред компютърните алгоритми да

се състоят от „изпълними стъпки”. [5]

и) **Ефективност**. „Алгоритмичният процес е ефективен, ако приключва в „реално”

време и всички присъщи му резултати се получават след „приемлив” брой стъпки”.

Алгоритми, които не решават задачата в ”разумен” срок от време или при

изпълнението им се налага съхраняване на твърде много начални или междинни

резултати, не са ефективни алгоритми в информатиката [6]

* Видове алгоритми:
  + Линейни
  + Алгоритьм за груба сила( Brute Force Algorithm)
  + Разклонени
  + Циклични
  + Рекурсивни (Recursive Algorithm)
  + Алгоритьмза обратно проследяване (Backtracking Algorithm)
  + Алгоритьм за тьрсене(Searching Algorithm)
  + Алгоритьм за сортиране (Sorting Algorithm)
  + Алгоритьм за хеширане (Hashing Algorithm)
  + Алгоритьм „Разделяй и владей"( Divide and Conquer Algorithm)
  + Евристични
  + "Алчни" алгоритми (Greedy Algorithm)
  + Вероятностни
  + Паралелни
  + Алгоритьм за динамично програмиране(Dynamic Programming Algorithm)
  + Рандомизиран алгоритьм (Randomized Algorithm)
* Предимства на алгоритмите
  + Лесно е за разбиране.
  + Алгоритьмът е поетапно представяне на решение на даден проблем.
  + В алгоритьма проблемът е разбит на по-малки части или стьпки, следователно

за програмиста е по-лесно да го преобразува в действителна програма.

* Недостатъци на алгоритмите
  + Писането на алгоритьм отнема много време.
  + Разбирането на сложна логика чрез алгоритми може да бъде много трудно.
  + Изявленията за разклоняване и цикъл са трудни за показване в Algorithms(imp).

1. Видове алгоритми за сортиране

* Алгоритъмът за сортиране е алгоритъм, съставен от поредица от инструкции, които приемат

масив като вход и извеждат сортиран масив.

* Има много алгоритми за сортиране, като например:

Selection Sort, Bubble Sort, Insertion Sort, Merge Sort,

Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket

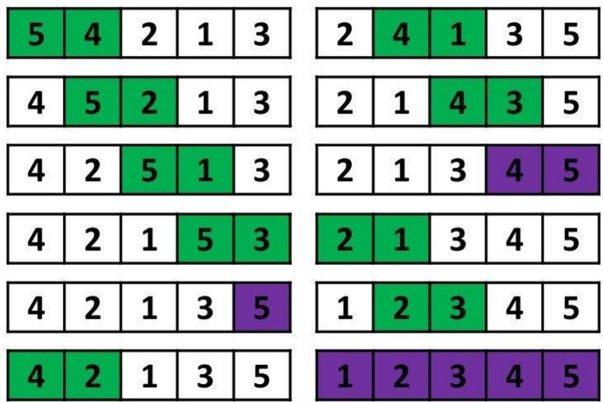
Sort, ShellSort, Comb Sort, Pigeonhole Sort, Cycle Sort

1. **Bubble Sort**

* **Bubble Sort** e най-простият алгоритъм за сортиране, който работи чрез многократна размяна на съседните елементи,

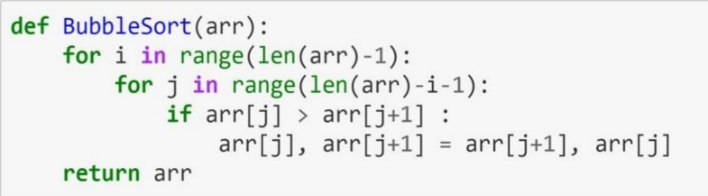
ако са в грешен ред.

* + Алгоритъм
* Стъпка 1: Сравнете всяка двойка съседни елементи в списъка
* Стъпка 2: Разменете два елемента, ако е необходимо
* Стъпка 3: Повторете този процес за всички елементи, докато целият масив бъде сортиран.

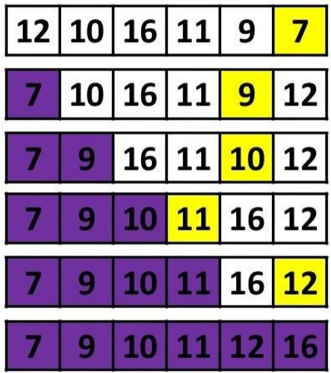


* + Времева сложност: О(n2),

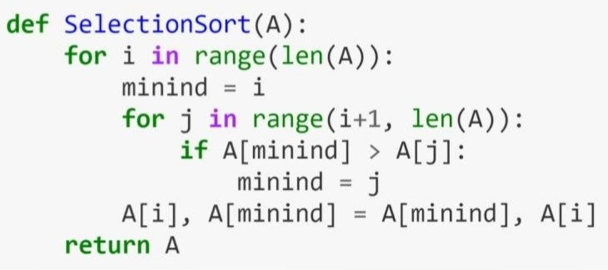
тъй като има два вложени цикъла



1. **Selection Sort**
   * **Selection Sort** e aлгоритъмът, който сортира масив, като многократно елемент (като се има предвид възходящ ред) от несортираната част и го поставя в началото. за сортиране намира на минималния



* + Алгоритъм:
    - Стъпка 1: Намерете минималната стойност в списъка
    - Стъпка 2: Разменете я със стойността в текущата позиция
    - Стъпка 3: Повторете този процес за всички елементи, докато целият масив бъде сортиран



* + Времева сложност: О(n2),

тъй като има два вложени цикъла

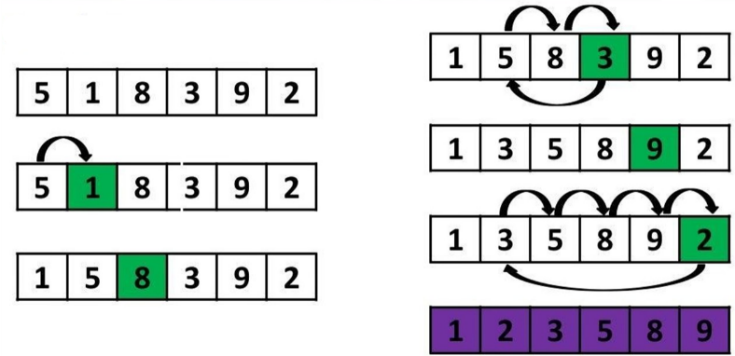
1. **Insertion Sort**

* **Insertion sort** e прост алгоритъм за сортиране, който работи по начина, по който сортираме картите за игра в ръцете си.
* Алгоритъм:
  + Стъпка 1: Сравняваме всяка двойка съседни елементи в списъка
  + Стъпка 2: Вмъкваме елемент в сортирания списък, докато заеме

правилното място

* + Стъпка 3: Разменяме два елемента, ако е необходимо
  + Стъпка 4: Повтаряме този процес за всички елементи, докато целият

масив бъде сортиран

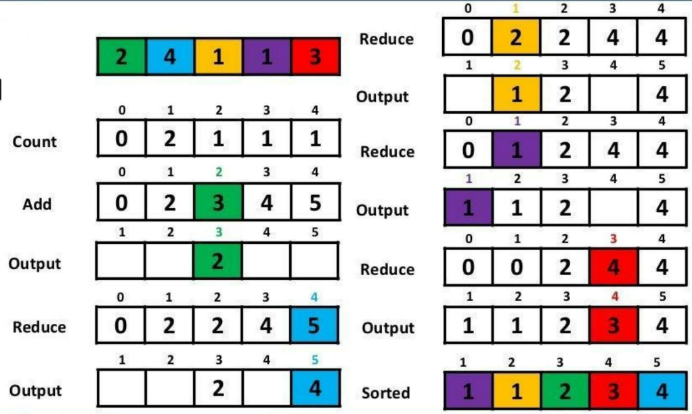


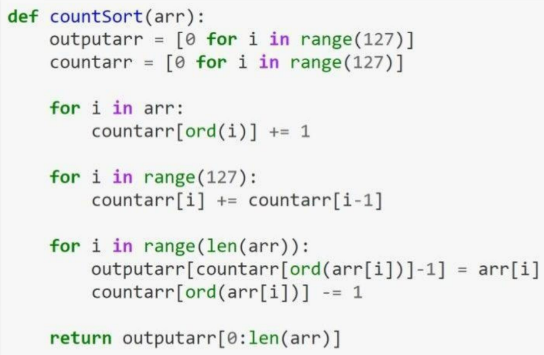
* + Времева сложност: О(n2),

тъй като има два вложени цикъла



1. **Counting Sort**

* **Counting sort** е техника за сортиране, базирана на ключове между конкретен диапазон. Работи, като преброява броя на обектите, които имат различни ключови стойности (вид хеширане). След това изчислява позицията на всеки обект в изходната последователност.
* Алгоритъм
  + Стъпка 1: Създайте масив за преброяване, за да съхранявате броя на всеки уникален обект
  + Стьпка 2: Променете масива за броене, като добавите предишното Число.
  + Стьпка 3: Създайте изходен масив чрез намаляване на масива за броене



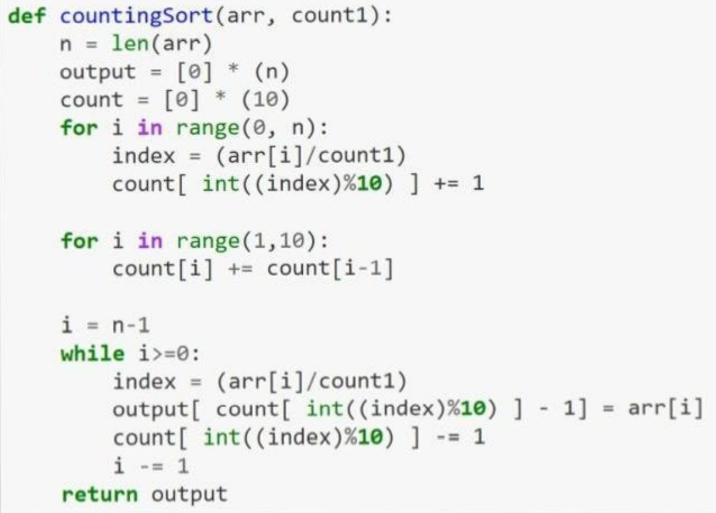
* Времева сложност: 0(n+k),

където "n" е броят на елементите във входния масив,

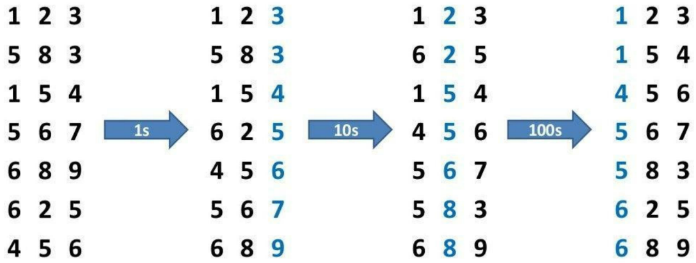
а "k" е диапазонът на входа.

1. **Radix Sort**

* **Radix sort** e алгоритьм, който сортира числа чрез обработка на цифри на всяко число, започвайки от най-малката цифра (least significant digit -LSD) или започвайки от най-значимата цифра (most significant digit - (MSD). Идеята на Radix Sort e да се извършва сортиране цифра по цифра, като се започне от най-малката цифра до най-значимата цифра. Radix Sort използва сортиране с броене като подпрограма за сортиране.
* Алгоритъм
  + Стьпка 1: Вземете най-малката цифра от всеки елемент
  + Стьпка 2: Сортирайте списъка с елементи въз основа на тази цифра
  + Стъпка 3: Повторете сортирането с всяка по-значима цифра

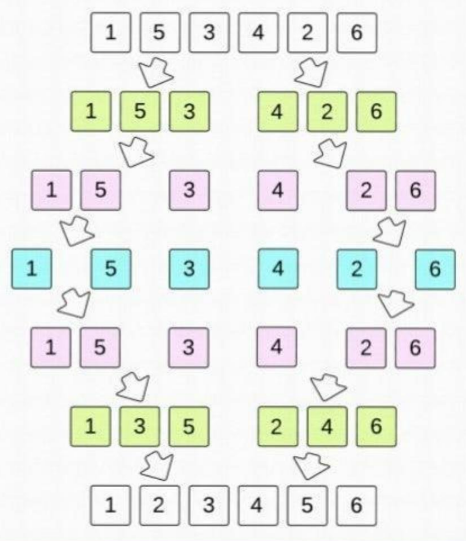


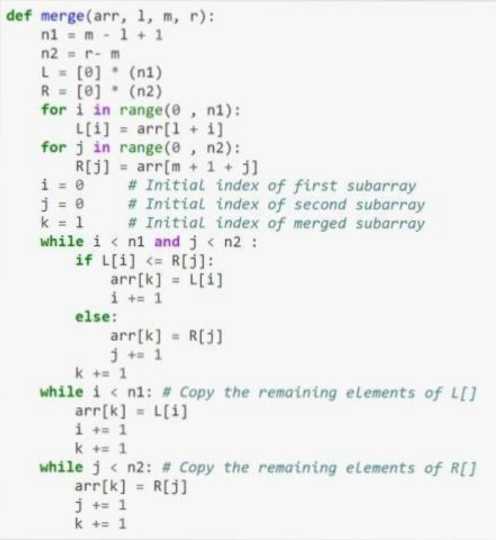
* Времева сложност: 0(n+k/d), където "n" е броят на елементите във входния масив, "k" е диапазонът на входа и "d" е броят на цифрите.



i

1. **Merge Sort**

* **Merge Sort** e алгоритьм за разделяне и владеене (Divide and Conquer). Той разделя входния масив на две половини, извиква себе си за двете половини и след това обединява двете сортирани половини.
* Алгоритъм:
  + Стъпка 1: Разделете списъка рекурсивно на две половини, докато вече не може да бъде разделен
  + Стъпка 2: Обединете (завладейте) по-малките списъци в нов списък в сортиран ред



* Времева сложност: 0(n \* log(n))

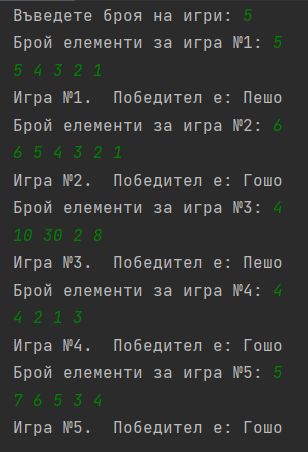
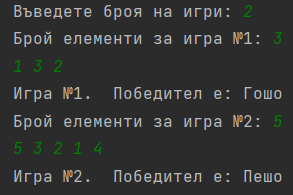
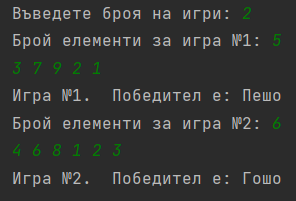


**Правила: Игра на Пермутация**

**Алгоритъм написан на Python**

def bubbleSort(inputArr):  
  
 n = len(inputArr)  
  
 # Проходи през всички елементи  
 for i in range(n):  
 # Последните i елемента вече са сортирани, затова можем да ги пропуснем  
 for j in range(0, n - i - 1):  
 # Сравняваме съседни двойки и разменяме ги, ако са в грешен ред  
 if inputArr[j] > inputArr[j + 1]:  
 inputArr[j], inputArr[j + 1] = inputArr[j + 1], inputArr[j]  
  
 return inputArr  
  
def countMovesSorted(inputArr):  
 # Функция, която сортира входния масив и връща броя на инверсиите (размени)  
  
 #Сортираме масива, за да знаем кога е във възходящ ред  
 copyArrSort = bubbleSort((inputArr.copy()))  
  
 #Променлива за ходовете на играчите  
 countMoves = 0  
  
 #Безкраен цикъл, докато не се подреди масива във възходящ ред  
 while(copyArrSort != inputArr):  
 #Взимане на индекса на най-големия елемент  
 max\_index1 = inputArr.index(max(inputArr))  
  
 #Премахване на най-големия елемент от оригиналната игра  
 inputArr.pop(max\_index1)  
  
 #премахване на най-големия елемент от сортираният елемент  
 copyArrSort.pop()  
  
 #Увеличаваме броя хода с единица  
 countMoves += 1  
  
 #Връщаме броя ходове  
 return countMoves  
  
  
def permutationGame(N, permutation):  
  
 for i in range(N):  
  
 countMove = countMovesSorted(permutation)  
  
 # Проверка: Кой е победил, като Гошо винаги е първи  
 if countMove % 2 == 1:  
 winner = "Гошо"  
 else:  
 winner = "Пешо"  
  
 return winner  
  
# Четене на броя на тестовите случаи  
gameCount = int(input("Въведете броя на игри: "))  
  
for n in range(gameCount):  
 # Четене на броя на елементите и пермутацията  
 elementsNumber = int(input(f"Брой елементи за игра №{n+1}: "))  
  
 # Четене на елементите на играта (пермутацията)  
 permutation = list(map(int, input().split()))  
  
 # Определяне на победителя и извеждане на резултата  
 winner = permutationGame(elementsNumber, permutation)  
  
 # Принтиране на победителя в дадената игра  
 print(f"Игра №{n+1}. Победител e: {winner}")

Примерни Входни и Изходни данни:



* **Функцията countMovesSorted връща броя на премахнатите елементи (ходове), които са необходими, за да се подредят елементите във възходящ ред. Използвам финкцията bubbleSort, за да сортирам масива без да променям оригиналния.**

Източници:

1. (Лекция)Тема 1: Понятие за алгоритъм. Интуитивно понятие за алгоритъм - свойства, видове, характеристики. Понятие за програма и езици за програмиране
2. (Упражнение) Sorting Algorithoms I
3. (Упражнение) Sorting Algorithoms II
4. https://www.geeksforgeeks.org/introduction-to-algorithms/