**Ülesanne 1: Lineaarotsing (Linear Search)**

1. Eraldi python failis olemas.
2. a) Aeg - lineaarne, n on massiivi pikkus. Algoritm läbib iga elemendi, et leida sihtelement või kuni saab aru, et elementi pole olemas.

b) Ruum - Konstantne. Vajalik on ainult paar muutujat.

1. Saab kasutada andmebaasides ja failisüsteemides. Mure kohaks on see, et suurte andmemahtude koral on see aeglane, sest ta peab kõik elemendid läbi käima ehk pigem ei sobi suurtele andmebaasidele, selleks on paremaid lahendusi.

**Ülesanne 2: Kahendotsingu (Binary Search) rakendamine ja analüüs**

1. Eraldi pythoni failis olemas.
2. Linear searchi puhul on ajakompleksus lineaarne, binary searchi puhul aga logaritmiline.
3. Näide: Suur sorteeritud nimekiri nt õppejõudude telefoninumbreid. Sellisel juhul on binary search kasulik, sest saame leida konkreetse numbri. Linear search nõuaks iga numbri läbi käimist, mis võtaks väga kaua aega.

**Ülesanne 3: Jump Search**

1. Jump search toimib sorteeritud masiivi puhul. N-ö hüppab kindlate sammude kaupa, et leida vahemik, kus meie otsitav element võiks asuda. Vahemiku leidmisel kasutatakse lineaarset otsingut, et täpne element üles leida.

Kood:

def jump\_search(arr, target):

n = len(arr)

step = int(n\*\*0.5) siin saab valida endale sobiv hüppesamm!

siit edasi peaks tegema prev = 0 ja siis laskma kui arr min step, n -1 on väiksem kui target siis prev = step ja lisame stepile juurde. kui ei leia elementi siis -1 vastuseks ja kui arr(i) = target siis return i ehk tagastame indeksi.

1. Linear = lineaarne, binary = logaritm, jump search = juureline? pole kindel kas see päris õige sõna aga √ n igatahes.
2. Jump search on efektiivsem suurte andmemahtude korral või kui andmeid pole vaja pidevalt sorteerida. Erinevalt binary searchist ei vaja jump search andmete pidevat sorteeritud hoidmist.

**Ülesanne 4: Kolmikotsing ja kahendotsing**

1. Ternary search toimib sarnaselt binary searchiga, aga jagab massiivi kahe asemel kolmeks. Eeldatakse, et andmed on järjestatud. Kasutab kahte jagamispunkti.
2. Ternary on logaritmiline, alus on 3 ehk log3 n.
3. Ternaryl on eelis suurte andmemahtude puhul, aga kui andmeid on vähem siis tegemist on ehk liiga keerulise algoritmiga ja võiks kasutada binary searchi.

Binary search on lihtsam ja mugavam kasutada, mistõttu võiks väiksemate andmebaaside puhul seda eelistada.

**Ülesanne 5: Otsingualgoritmide praktiline rakendamine**

Valisin probleemiks - Saadaolevate lendude leidmine lennufirma broneerimissüsteemis.

Stsenaarium: Lennufirma broneerimissüsteemis on vaja kiiresti üles leida saadaval olevaid lende vastavalt kliendi nõudmistele nagu näiteks - kuupäev, sihtkoht, väljumiskoht jms.

Algoritmiks peaks valima kas Binary searchi või Ternary searchi, sest andmed on sorteeritud kas siis kuupäevade või sihtkohtade järgi. Lendude andmed on samuti järjestatud ehk siin on logaritmiline ajakompleksus, mis teeks omakorda just need algoritmid kõige kiiremaks ja efektiivsemaks. Lisaks kuna klient valides kuupäeva ja sihtkoha, välistab suure osa lende siis on tegemist väikese andmemahuga ja sellisel juhul töötavad need algoritmid kiiresti.

**Modifikatsioonid**

Lisafiltrid: Näiteks lennuklass, otselend või ümberistumisega, erivajadused jms.

Reaalajas värskendamine: Sellisel juhul peaks kaaluma algoritmi mis toetuks dünaamiliselt muutuvale andmestruktuurile, mis on veidi keerulisem.

**Boonusülesanne:**

Fibonacci search kasutab fibonacci jada ja suhet, et määrata otsitava elemendi ligikaudne asukoht sorteeritud listis. Algoritm ise on sarnanae binary searchiga, aga jagab massiivi kahe asemel fibonacci jada arvu järgi. Kui tahame seda kasutada suuremahuliste andmete puhul siis peaks alustama sellega, et jagame need kahte ossa. Seejärel määrab algoritm suhtest lähtuvalt jagamis punkti, et vähendada andmete arvu mida kontrollida. Algoritm jätkab jagamist ja kontrollimist kuni leiab otsitava elemendi või kui terve asi läbi käidud ja elementi pole.