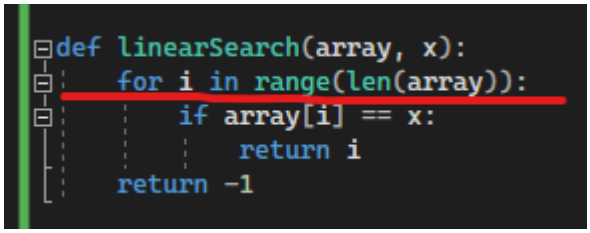
Kodutöö 3

Otsingualgoritmide analüüs

Ülesanne 1: Lineaarotsing

Lineaarotsingu ajakompleksus on O(n), kuna töötleme ainult ühes tsüklis terve listi läbi. Parim oleks O(1), kui otsitav arv on esimesel positsioonil.

Ruumikompleksus on O(1), kuna otsingus kasutame võrdlemiseks ühte muutujat „i".

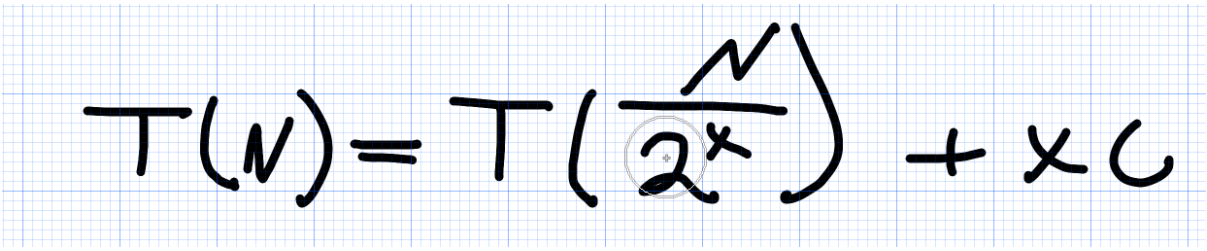


Reaalmaailmas saaks kasutada lineaarotsingut, et teada saada mitmendal kohal mingi objekt järjekorras on. Näiteks tuludeklaratsiooni esitamise ajal on serverid üle koormatud ja sisselogimine on aeglustunud ning inimesed näevad mis kohal nad on järjekorras sisselogimisel.

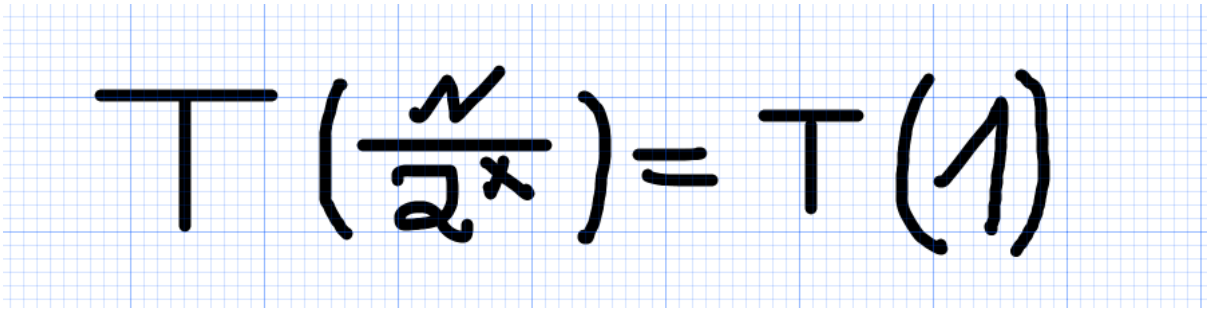
Ülesanne 2: Binary search

Otsime elementi, mida ei ole loendis ehk peame läbi käima terve loendi.

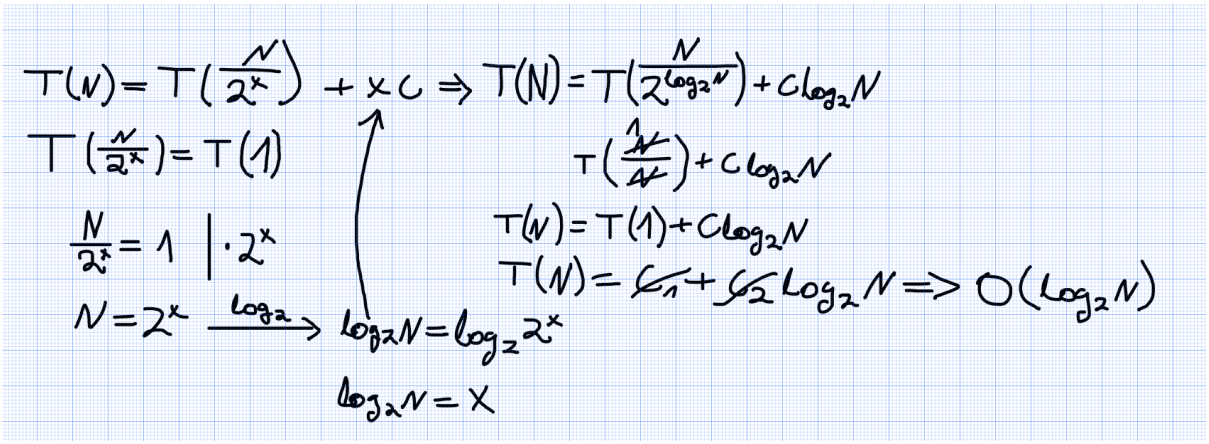
Otsingu muster on $N/2^x$, kuna iga iteratsiooniga me poolitame otsimisruumi.



Viimane iteratsioon on valimist 1 kuna oleme kõik valikud kõrvaldanud.



T(1) võtab aega c1 kuna võrdlus ei sõltu mingist muust tegurist ning meil on võimalik leida binaarotsingu ajakompleksus milleks on O(logN).

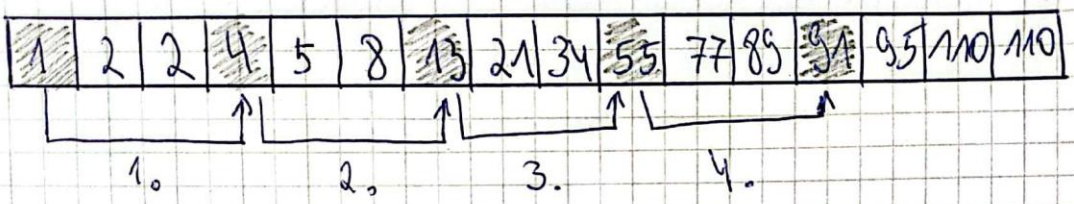


Võrreldes lineaarotsingu ja binaarotsingu ajakompleksust, siis binaarotsingul on eelis suurematel otsingutel.

Üks eelis lineaarotsingul binaarotsingu üle on, et loend võib olla juhuslikus järjekorras, aga binaarotsing nõuab, et sisend oleks sorteeritud. Sel juhul kui on antud suvaline andmetabel, mis ei ole sorteeritud, tasuks kasutada lineaarotsingut, et leida kas otsitav element on tabelis või mitte.

Ülesanne 3: Jump search

Jump search



loendi suurus n = 16

sammu suurus m = $\sqrt{n}$ = $\sqrt{16}$ = 4

Soovime leida arvu 77

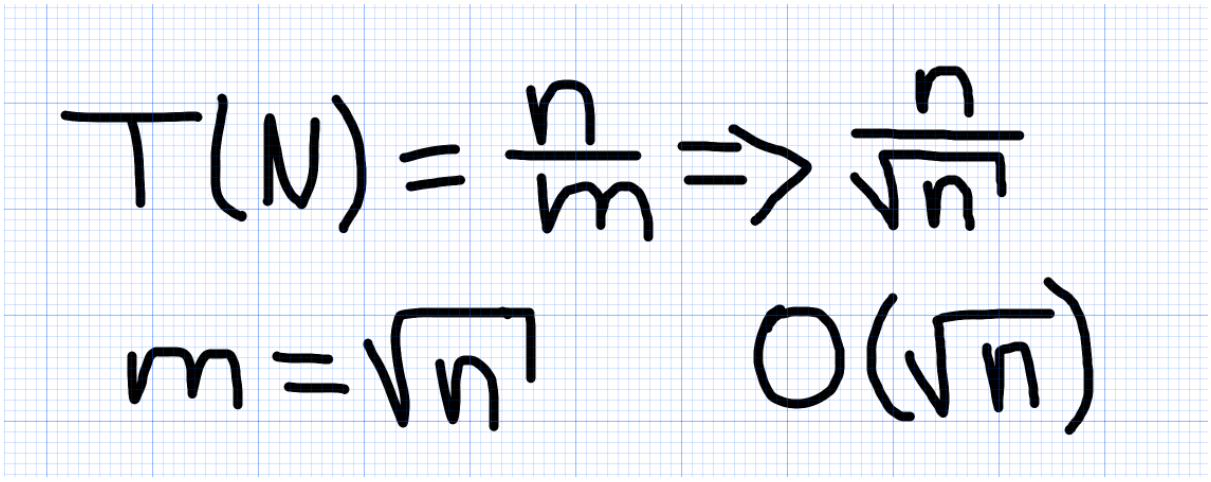
Astume 4 sammu kuniks leiame elemendi mis on suurem või võrdne otsitava arvuga.

Alustame alt loendi algusest.

4. Kuna 91 > 77, siis teeme lineaarotsingu tagurpidi kuniks leiame otsitava arvu.

```
jumpSearch(loend, x, n)
    m = sqrt(n)
    # eelmine_viimane => el_v = 0
    while loend[min(samm, n)] < x:
        el_v = samm
        samm += sqrt(n)
    while loend[el_v] < x:
        el_v += 1
    if loend[el_v] == x:
        return el_v
```

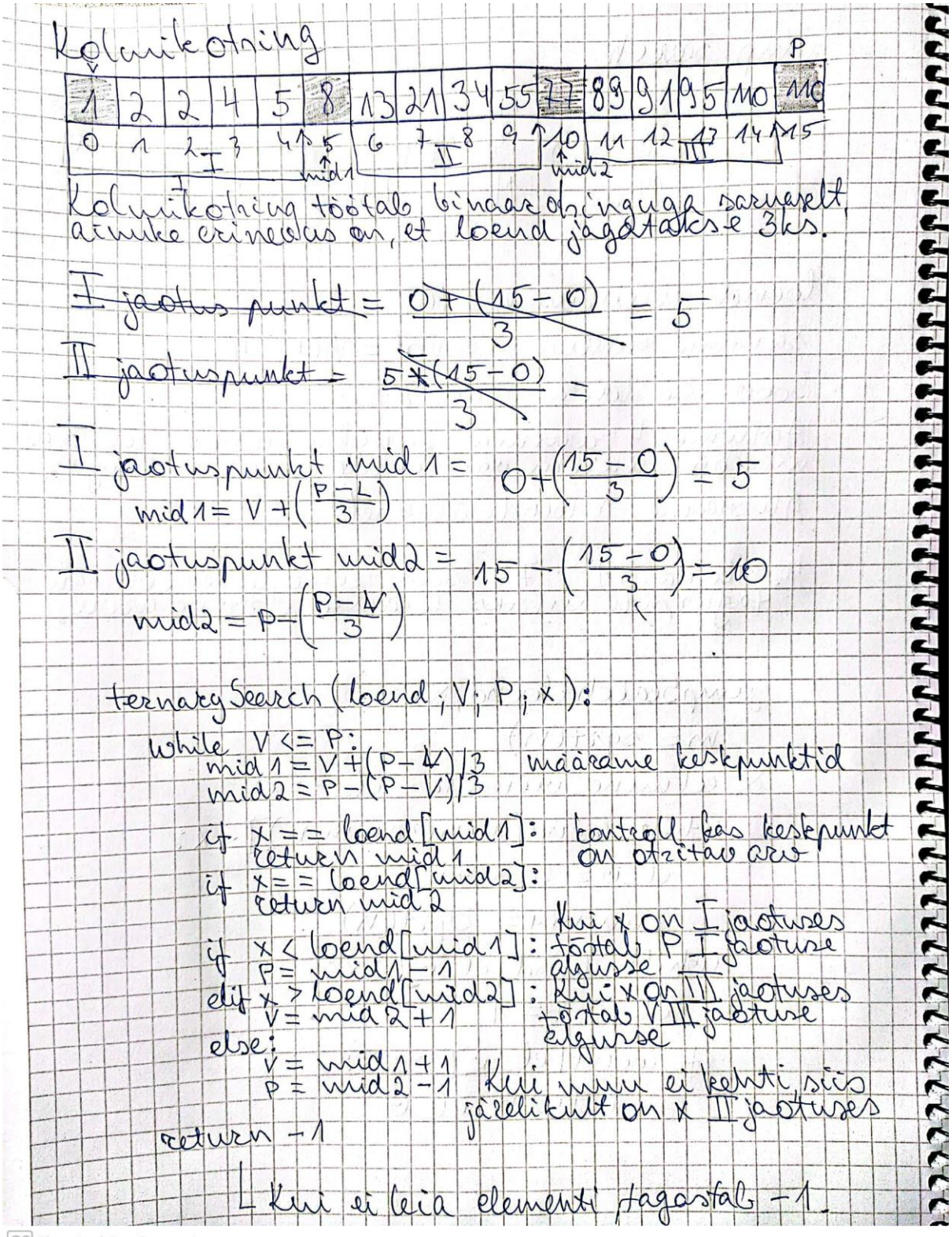
Optimaalne sammude arv on ruutjuur loendi pikkusest ning igal iteratsioonil kasvab samm m korda. Algoritmi ajakompleksus on O(√n).



StackOverflow foorumist leidisn selgituse, kus Jump Searchil võib olla eelis lineaar- ja binaarotsingu üle. Kui info on talletatud näiteks kasseti peal, kus tagasiliikumine on kulukas, sest kassett tuleb peatada, tagasi kerida ning siis uuesti läbi käia. Jump Searchiga avastaks otsitava elemendi kiiremini kuna teame, mis hetkel vaja tagasi kerida.

https://stackoverflow.com/questions/58472462/in-binary-search-why-does-traversing-back-cost-more-than-traversing-forward

Ülesanne 4: Kolmikotsing ja Kahendotsing



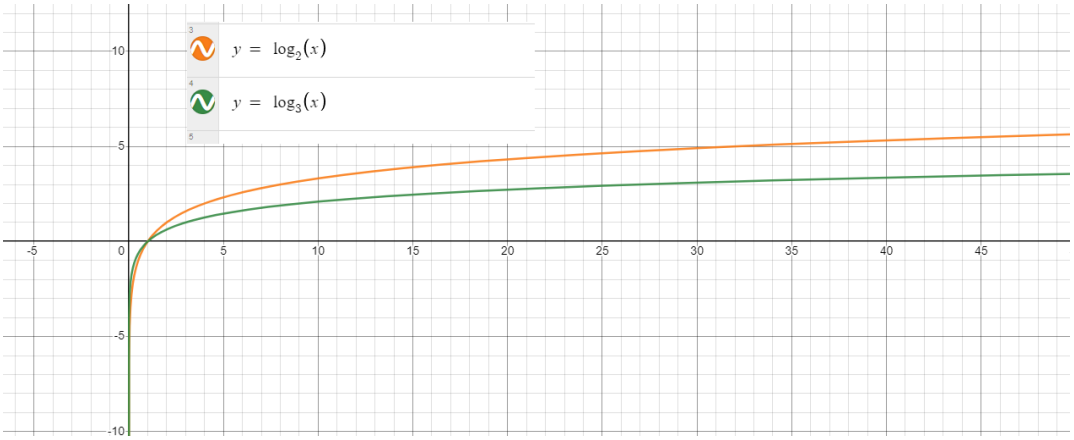
On teada, et binaarotsingu ajakompleksus on $0(\log_2 N)$. Kolmikotsingu ajakompleksus on $0(\log_3 N)$.

TIME COMPLEXITY

ALGORITHM	BEST CASE	AVERAGE CASE	WORST CASE
LINEAR SEARCH	O(1)	O(N)	O(N)
BINARY SEARCH	O(1)	O(LOG2N)	O(LOG2N)
TERNARY SEARCH	O(1)	O(LOG3N)	O(LOG3N)
JUMP SEARCH	O(1)	O(√N)	O(√N)
INTERPOLATION SEARCH	O(1)	O(LOG(LOG(N)))	O(N)

Joonis 1 https://www.codingninjas.com/studio/library/time-space-complexity-of-searching-algorithms

Graafikul ütleks, et kolmikotsingul on eelis binaarotsingu üle.



Kuigi mõlemad algoritmid töötavad sarnaselt teeb halvimal juhul kolmikotsing ikkagi rohkem võrdlusi kui binaarotsing.

```
def binaar_otsing(arr, x):
    low = 0
    high = len(arr) - 1
    mid = 0

    while low <= high:
        mid = (high + low) // 2

        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid
    return -1
```

C1 C2

```
while V <= P:
mid1 = V + (P-V)/3
mid2 = P - (P-V)/3
C1 if x == loend[mid1]:
       return mid1
C2 if x == loend[mid2]:
       return mid2
C3 if x < loend[mid1]:
       P = mid1 - 1
C4 elif x > loend[mid2]:
       V = mid2 + 1
   else:
       V = mid1 + 1
       P = mid2 - 1
```

Ülesanne 5: Otsingualgoritmide praktiline rakendamine

Kiire kontakti leidmine telefonis

Eeldame, et kontaktid on telefoni salvestatud korrektselt. Kontaktil on ees- ja perenimi ning järjestatud tähestikulises järjekorras. Ainult numbrina salvestatud kontakte meil ei ole.

Arvan, et kõige tõhusam algoritm oleks binaarotsing, sest olenemata kui suur on kontaktinimekiri, teeb binaarotsing uuritud otsingualgoritmidest kõige vähem võrdlusi võrreldes teistega.