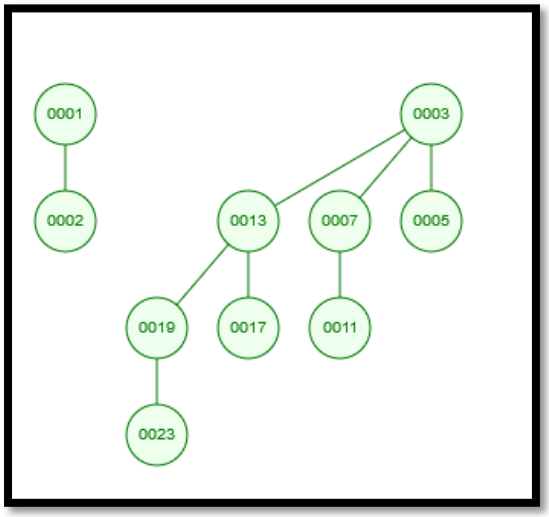
Naloga 3.1

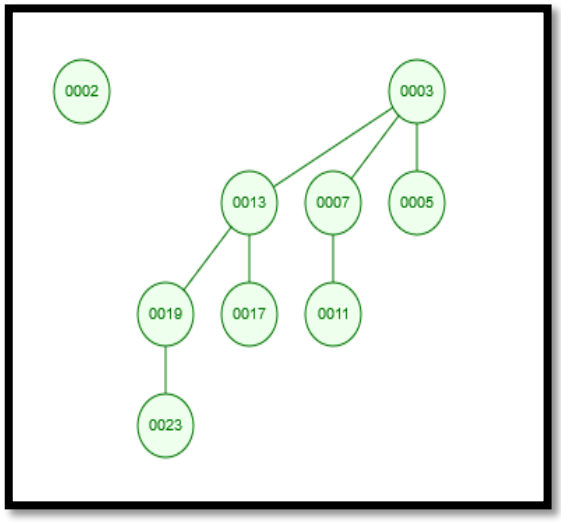
A)

a) Imejmo binomsko minimalno-urejeno kopico. Izvedite naslednje operacije Insert(23), Insert(19), Insert(17), Insert(13), Insert(11), Insert(7), Insert(5), Insert(3), Insert(2), ***Insert(1)****,* ***DeleteMin()****,* ***DecreaseKey(23, 1)****,* ***DeleteMin()****. Narišite stanje podatkovne strukture po koncu vsake poudarjene operacije.*

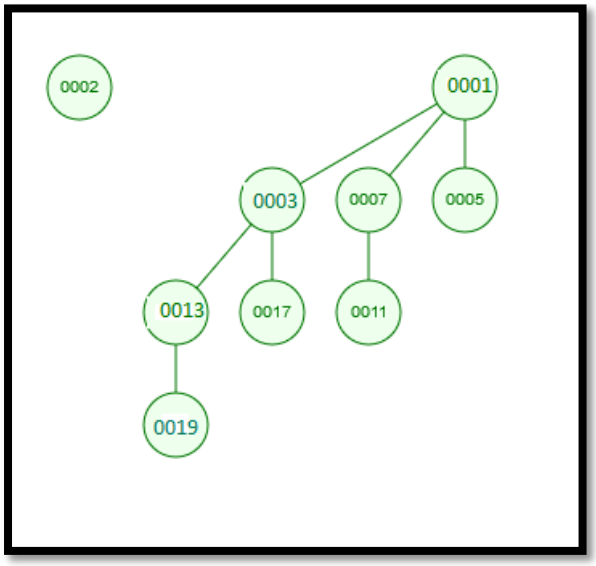
Insert(1)



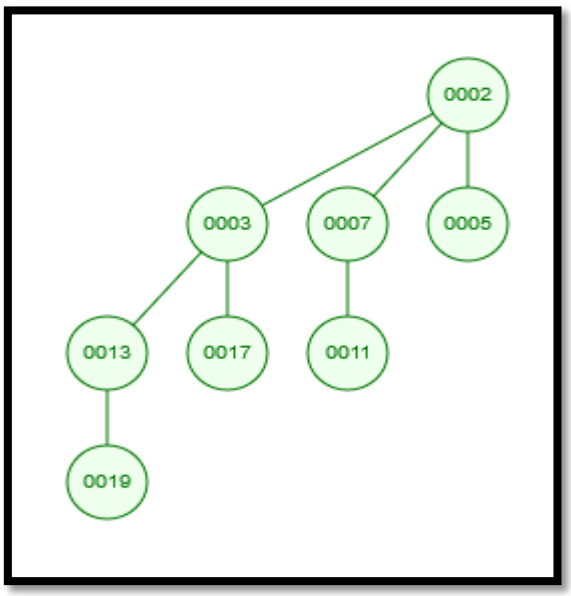
DeleteMin()



DecreaseKey(23,1),

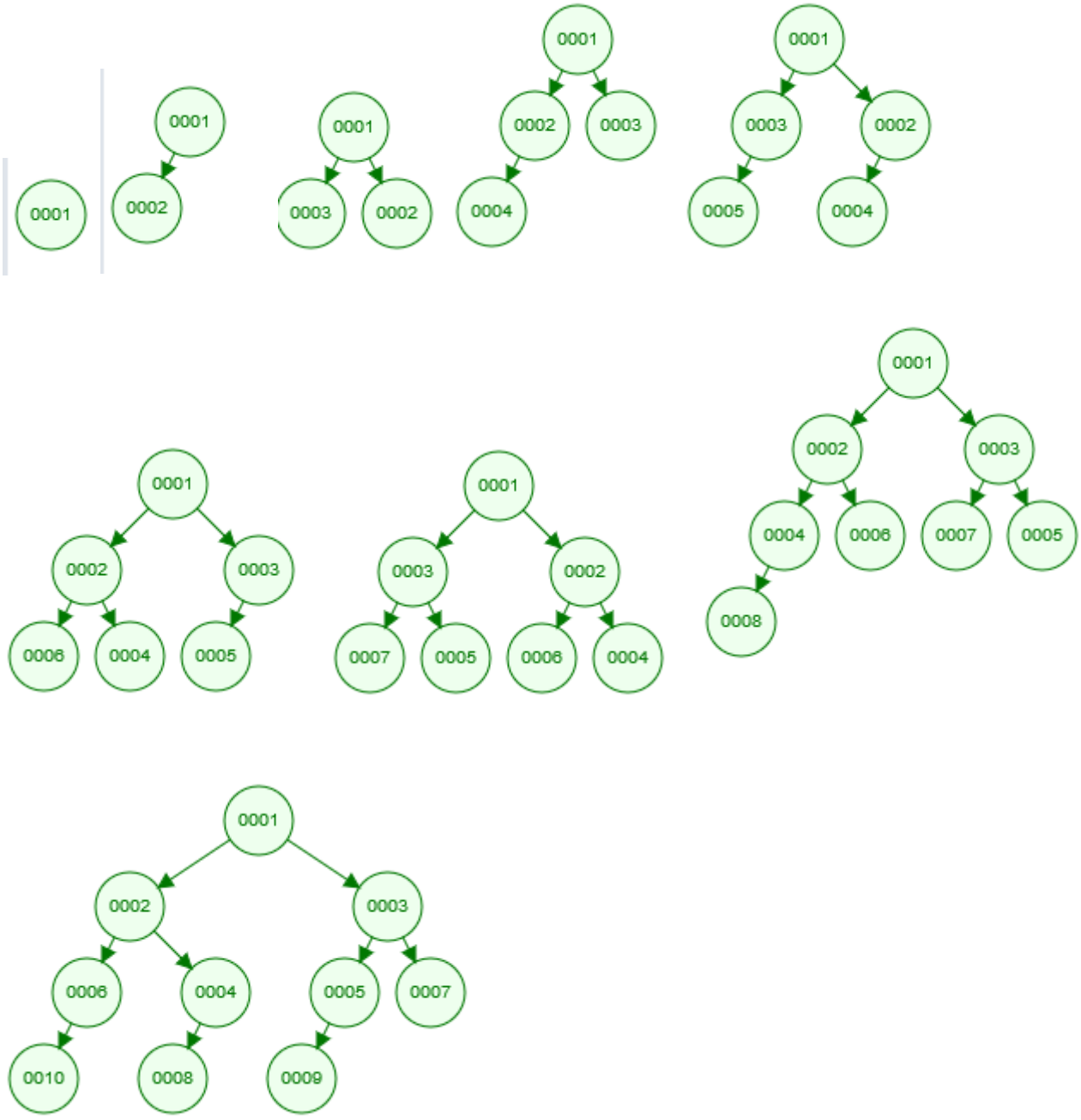


DeleteMin()



B)

V literaturi poiščite opis skrivljene kopice (angl. skew heap). Opišite, kako izgleda operacija zlivanja dveh skrivljenih kopic. V prazno omenjeno kopico vstavite elemente {1...8}. Ob koncu vsakega vstavljanja narišite sliko kopice. Za dodatne točke, narišite kopico po tem, ko vanjo vstavite po vrsti elemente {1, 2, 3, ..., n}.



Najprej primerjamo korene obeh kopic.

Koncni koren je koren kopice ki ima manjsi koren in koncno desno poddrevo je levo poddrevo kopice z manjsim korenom.

Za konec pa še primerjamo kočno levo poddrevo tako da rekurzivno združujemo desno poddrevo kopice z večjim korenom.

c)

V literaturi poiščite opis kopice min-maks (angl. min-max heap). V čem se razlikuje od običajne dvojiške kopice, katere operacije omogoča in kakšna je njihova časovna zahtevnost? V poljubnem programskem jeziku napišite funkcijo `boolean getLayer(int i)`*, ki z uporabo osnovnih aritmetičnih operacij (+, -, *, /, %, <<, >>), primerjavami (<, >, ==) in kontrolnimi stavki (if, while) za podan indeks vozlišča izračuna, ali je vozlišče na min (0) ali maks (1) nivoju. Koliko je časovna zahtevnost vaše funkcije v odvisnosti od podanega števila?*

Min-max kopica je popolna dvojiška struktura ki pa ubistvu vzame prednosti min kopice ina pa tudi max kopice Dobimo konstanni čas vračanja in logaritmični čas Brisanja. Vsak node in sodo stopnji je manjši kot pa vsi njegovi potomci dokler vsak node na lihi stopnji je pa večji od vseh svojih potomcev

```
V kopico vstavljamo kot pri običajni kopici,
Ko pa vstavimo x pogledamo:
*if  na max nivoju: check x < parent;
     Če je ga zamenja
*if na min nivoju: x > parent;
     Če je ga zamenja
To pa se nadaljuje rekurzivno do roota.
```

Naloga 3.2

Skupina n otrok se postavi v vrsto in se igra izštevanko. Če ima izštevanka ima m zlogov, otrok na m-tem mestu izpade in izštevanka se ponovno začne pri otroku m + 1. Če pridemo do konca vrste, začnemo ponovno od začetka. Izštevanko ponavljamo toliko časa, dokler ne izpade zadnji otrok. Zanima nas vrstni red izpadanja otrok glede na njihovo zaporedno številko v začetni vrsti. Na primer, če ima izštevanka m=3 zloge in imamo n=10 otrok, potem izpade najprej 3. otrok, potem 6., 9., 2., 7. itd.

A) *Zapišite vrstni red izpadanja za m=6, n=20*

Vrstni red je: 6 12 18 4 11 19 7 15 3 14 5 17 10 8 2 9 16 13 1 20.

B) *Napišite psevdokodo čimbolj učinkovitega algoritma izštevanja, ki vrne zaporedje izpadlih za poljubni m in n. Ocenite časovno in prostorsko zahtevnost algoritma glede na n in m.*

```
Function izloči(int m,int n){
      Int[] Otroci = new int[n];
      Int[] izloceni = new int[n];
      m-=1;
      int j = 0;
      i = m;
      while(otroci.length() > 1){
            izloceni[j] = otroci[i];
            i = (m+i)%otroci.length();
            j++;
      }
      Izloceni[j]=otroci[0];
      Return izloceni;
}
```

Časovna Zahtevnost je O(n) ker moramo čez celo tabelo otrok z 1 zanko prostorska pa tudi O(2n) ker nakoncu imamo 2 tabeli po 2n elementov

C) *Sedaj spremenimo pravila igre: izpadli otrok izbere izštevanko za naslednje izštevanje (torej izbere m). Napišite psevdokodo čimbolj učinkovitega algoritma izštevanja za dani n in spremenljivi m. Ocenite časovno in prostorsko zahtevnost algoritma v odvisnosti od n in m.*

```
Function izloči(int m,int n){
      Int[] Otroci = new int[n];
      Int[] izloceni = new int[n];
      m-=1;
      int j = 0;
      i = m;
      while(otroci.length() > 1){
            izloceni[j] = otroci[i];
            m = random.randInt(0,10);
            i = (m+i)%otroci.length();
            j++;
      }
      Izloceni[j]=otroci[0];
      Return izloceni;
}
```

Vse je isto samo da vsakič posebaj m nastavimo na novo. To sem kar impementiral s fucnkcijo random ki ponazarja random odločitev otroka. Časovna zahtevnost je O(n) ker moramo iti čez vse otroke v vsakem primeru. Prostorka je pa O(2n) ker imamo nakoncu 2 tabeli po n elementov

Naloga 3.3

Upam da ni izbrana ta naloga ☺

Peter Zmeda rešuje problem rezanja jeklenih palic v železarni na Jesenicah. Palico celoštevilske dolžine *n* dm je mogoče razrezati na dele celoštevilske dolžine. Nekje je slišal, da bo za problem lahko uporabil pristope dinamičnega programiranja.

a) *Cene različnih dolžin palic na na trgu so:*

dolžina (dm)	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
cena (EUR)	*2*	*4*	*7*	*9*	*10*	*11*	*12*	*19*	*22*

Nekaj podobnega smo delali na vajah

Dolžina	Cena	Dolžina rezov
7	16	3,3,1
8	19	8
9	22	9

b) *Razmere na trgu delovne sile so se spremenile. Pomagajte Petru sestaviti algoritem, če se cene palic na trgu sicer niso spremenile, le:*

1. *za vsak rez palice moramo odšteti 2 EUR; ali pa*
2. *za vsak rez palice moramo odšteti b/a EUR, kjer je b/a razmerje nastalih dolžin ko razrežemo palico dolžine n, ter je a krajša od nastalih dolžin (torej,* $n=a+b$ *in* $a \leq b$ *).*

Za vsakega od obeh primerov podajte cenovno funkcijo ter psevdokodo za izračun optimalnega razreza in iztržene cene. Ciljana časovna zahtevnost algoritmov je $O(n_2)$.

i .)

Cenovna funkcija za palico dolžine i = palice[j].optimalna + palice[i - j].optimalna – 2
Za lažji izračun sem si shranil tudi palico 0. Cene so nastavljene na dejansko ceno palice z dolžino n.
Optimanla bo na začetku in in se potem nakanadno spreminja.

```
For(i=0; i<#palic;i++){
        For(j=i-1; j<i/2;j++)
        If(cena izračunana v tem koraku > max_value)
                Spremenimo ceno palice

Pogledamoše če je cena dražja od optimalne ki smo jo izačunali če je jo zamenjamo;
}
```

ii.)

Cena za palico dolžine I == palica[j].optimala + palica[i-j].optimalna – palica[j].optimalna/palica[i-j].optimalna

Tako napišemo:

```
For(i=0; i<#palic;i++){
        For(j=i-1; j<i/2;j++)
        If(cena izračunana v tem koraku > max_value)
                Spremenimo ceno palice

Pogledamoše če je cena dražja od optimalne ki smo jo izačunali če je jo zamenjamo;
}
```

C.) *Kriza na trgu se je še poglobila in zato ima Peter na voljo računalnik z omejenim pomnilnikom. Pri računu si lahko zapomni zgolj k rešitev podproblemov, kjer je (k < n), ostale podprobleme pa bo moral računati sproti oziroma ponovno. Pri tem predpostavimo, da globina sklada za rekurzivne klice ni omejena.*

Peter bi si sporti shranjeval že izračunane probleme in naenkrat računal samo za dejanski problem, tako bi prihranili ogromno časa za računanje. Vendar bi bilo mogoče pametno omejiti do kam lahko gre rekurzija ker drugače lahko dosežemo mejo pomnilnika in bo vse nehalo delovati