Några svar till exempeltenta i TDDD86 Datastrukturer, algoritmer och programmeringsparadigm

Följande är lösningsskisser och svar till uppgifterna på exempeltentan. Lösningarna som ges här ska bara ses som vägledning och är oftast inte tillräckliga som svar på tentan.

2. 1 (D), linjär.

   2 (G). Varje iteration av den inre loopen är kvadratisk i den yttre loop-variabeln. Ett enkelt sätt att komma fram till resultatet är att inse att vi vill ha summan $\sum_{i=0}^{n-1} i^2$.

   3 (H). Varje anrop ger upphov till två nya anrop. När vi kommer till basfallen (där $n = 1$) har vi genererat 2^n anrop till `f3`

   4 (D). Det här fallet är likt Mergesort och Quicksort, förutom att varje rekursivt anrop bara utför en konstant mängd arbete i stället för en linjär mängd arbete. Mönstret är samma som det för att bygga en heap från botten och upp. Längst upp gör vi 1 enhet arbete, på andra nivån gör vi 2 enheter arbete, på tredje nivån 4 enheter, etc. Den totala mängden arbete ges därför av $1 + 2 + 4 + 8 + \ldots + n$. Denna summa är linjär i n.

   5 (E). Det här är exakt samma mönster som i Mergsort och Quicksort. Om man vill tänka på det som en summa så blir det $n + n + \ldots + n$ där det finns $\log_2 n$ summander.

   6 (B). Efter första iterationen blir $i = 2$. Efter andra blir $i = 2^2$. Efter tredje blir $i = 2^{2^2}$, etc. Den här proceduren använder $\log^* n$ steg för att nå n. Det går bra att hänvisa till att $\log^* n$ var det enda möjliga svaret med en tillväxthastighet mellan konstant och logaritmisk.

3. Huvudidén är att använda en ordnad symboltabell för att lagra strängarna på stacken, där den i:te strängen som satts in är värdet associerat med heltalsnyckeln i. Om symboltabellen implementeras med t.ex. ett AVL-träd kan vi lätt hitta och ta bort strängen som senast sattes in i stacken (dess nyckel är den största i trädet). Vi kan också enkelt slumpa fram ett heltal och snabbt ta bort motsvarande sträng.

```
class LeakyStack {
public:
  void push(string item) {
     st.insert(counter++, item);
  }

  string pop() {
    string item = st.max();
    st.deleteMax();
    return item;
  }
```

```
  void leak() {
    int r = Random.uniform(st.size());
    st.delete(r);
  }

private:
  int counter = 0;
  AVLTree<int, string> st;
};
```

4. 1. Sortera varje rad med heap-sort.
   2. For varje tal i rad 0, från störst till minst, använd binärsökning för att kontrollera om det uppträder på de andra $N - 1$ raderna.
   3. Returnera det första talet som uppträder på alla N rader.

   Exekveringstiden växer som $N^2 \log(N)$ med flaskhalsar i steg 1 och 2. Korrekthet följer eftersom det största talet måste finnas på rad 0. Undersökningen av rad 0 från störst till minst försäkrar att vi hittar det *största* gemensamma talet.