HEIMADÆMI 5

TÖL203G Tölvunarfræði 2

Kári Hlynsson[1]

Háskóli Íslands

23. febrúar 2023

Verkefni 1

Skoðið Java kóðann fyrir Mergesort (ALGORITHM 2.4 í bókinni, glæra 8 í fyrirlestri 9). Segjum að við köllum aðeins á `merge`-fallið (neðsta línan í `sort`-fallinu) ef `a[mid+1]` er minna en `a[mid]`.

(a) Hvers vegna er í lagi að gera þetta?

(b) Á hvernig inntaki myndum við græða mest á að bæta þessu inn?

Lausn

Hluti (a)

Hlutverk `merge` fallsins er að raða hlutfylkjum sem hefur þegar verið raðað endurkvæmt. Segjum sem svo að $\texttt{a[0]} < \cdots < \texttt{a[mid]}$ og $\texttt{a[mid+1]} < \cdots < \texttt{a[n]}$. Ef $\texttt{a[mid]} < \texttt{a[mid+1]}$.

Hluti (b)

Við græðum mest á þessari útfærslu ef sérhvert stak í fylkinu er í réttu hlutfylki. Til að sjá þetta betur skulum við taka dæmi. Látum `a[] = [0,4,3,2,1, 8,9,6,5,7]`. Þegar við köllum á `sort` byrjum við á því að sortera vinstri og hægri hlutann og fáum hlutfylkin `a[0..4] = [0,1,2,3,4]` og `a[5..9] = [5,6,7,8,9]`. Nú er `a[mid] < a[mid+1]` og við köllum ekki á `merge` en við sjáum jafnframt að fylkið er raðað svo það er óþarft.

[1] Slóð á Github kóða: `https://github.com/lvthnn/TOL203G/tree/master/HD5`

Verkefni 2

Leysið dæmi 2.3.18 á bls. 305 í kennslubókinni. Kallið ykkur útgáfu `QuickX` og berið hana saman við upphaflegu útgáfuna í bókinni (`Quick.java`) með forritinu `SortCompare.java`. Þið getið hent út úr `SortCompare` notkun á öðrum röðunaraðferðum. Skilið breyttu útgáfunni af `partition`-fallinu og niðurstöðu úr samanburði á `Quick` og `QuickX` í `SortCompare`. Til að fá raunhæfan samanburð notið `n = 1000000` og `trials = 10`. Þá ættuð þið líka að breyta útprentunarskipuninni í `SortCompare`, þannig að þið fáið fleiri aukastafi í hlutfallinu milli tímanna. Það ætti ekki að vera mjög mikill munur á þessum tveimur útgáfum. Hvers vegna?

Lausn

Útfærslan er til sýnis fyrir neðan í FORRITI 1. `median` er fall sem var útfært til að skila miðgildi þriggja staka. Úr því að `a[]` er stokkað í `public` útfærslunni af `sort` notum við fyrsta stakið í hlutfylkinu, miðjustak og síðasta stak.

```
18  private static int partition(Comparable[] a, int lo, int hi) {
19    int i = lo, j = hi+1;
20
21    int m = median(a, lo, (lo+hi)/2, hi);
22    Comparable v = a[m];
23    exch(a, lo, m);
24
25    while (true) {
26      while (less(a[++i], v)) if (i == hi) break;
27      while (less(v, a[--j])) if (j == lo) break;
28      if (i >= j) break;
29      exch(a, i, j);
30    }
31
32    exch(a, lo, j);
33    return j;
34  }
```

FORRIT 1: Útfærslan á `partition` í `QuickX` klasanum

Munur í tímaflækju á `Quick` og `QuickX` er nánast enginn. Meðaltal 10 mælinga úr `SortCompare` var 1.0036 (staðalfrávik 0.0659) svo keyrslutími er nokkurn veginn sá sami. Munurinn er lítill því `a[]` hefur þegar verið slembistokkað og við fáum því ekki nauðsynlega gott (miðlægt) vendistak.

Verkefni 3

Fylkinu `[2,3,1,4,5,7,6,8]` hefur verið skipt upp (*partitioned*) um vendistak (og það sett á réttan stað), en það er ekki gefið upp hvert vendistakið var. Hver af stökunum gætu hafa verið vendistakið? Teljið upp öll möguleg stök og rökstyðjið að þau séu möguleg og hin séu það ekki.

Lausn

Við getum skoðað hvort stak sé hugsanlegt vendistak út frá eftirfarandi: ef hlutfylkið vinstra megin við stakið inniheldur bara gildi minni en eða jöfn skiptistakinu, og hægra hlutfylkið bara gildi stærri en eða jöfn því, þá er það gilt. Athugum að þetta getur átt við um endapunkta fylkisins, ef þeir eru stærsta/minnsta gildið í fylkinu. T.d. getur `8` hafa verið skiptistak en þá endar það aftast í fylkinu því þar bendir `j` í lokin. Þetta er jafnframt stærsta stakið í fylkinu og uppfyllir að öll stökin í vinstra hlutfylkinu eru minna en (eða jafnt og) stakið.

Einnig eru `4` og `5` gild skiptistök, því þau uppfylla skilyrðið hér að ofan. Við sjáum aftur á móti að engin önnur stök í fylkinu gætu hafa verið skiptistak því það má finna stak í vinstra eða hægri hlutfylki þeirra sem brýtur þetta skilyrði. Tökum þetta saman í TÖFLU 1 hér fyrir neðan.

`i`	`a[i]`	Vendistak?	Athugasemdir
`0`	`2`	✗	`1` í hægra hlutfylki, $<$ `2`
`1`	`3`	✗	`1` í hægra hlutfylki, $<$ `3`
`2`	`1`	✗	sjá athugasemdir fyrir ofan
`3`	`4`	✓	
`4`	`5`	✓	
`5`	`7`	✗	`6` í hægra hlutfylki $<$ `7`
`6`	`6`	✗	`7` í vinstra hlutfylki $>$ `6`
`7`	`8`	✓	

TAFLA 1: Hugsanleg vendistök fyrir Quicksort

Verkefni 4

Gefið er fylkið `[5,4,8,1,6,3,5,7]`. Sýnið hvernig það raðast með hrúgurröðun (*heap sort*) með því að búa til mynd sem er sambærileg við myndina á bls. 324 í kennslubókinni (eða á glæru 35 í fyrirlestri 11). Sýnið líka hrúguna sem tvíundartré eftir að fylkinu hefur verið hrúguraðað (þrep 1).

Lausn

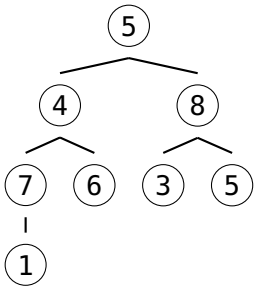
TAFLA 2 fyrir neðan sýnir keyrsluna á Heapsort og hvernig `a[]` breytist.

		a[i]								
N	k	0	1	2	3	4	5	6	7	8
upphaf			5	4	8	1	6	3	5	7
8	4		5	4	8	7	6	3	5	1
8	3		5	4	8	7	6	3	5	1
8	2		5	7	8	4	6	3	5	1
8	1		8	7	5	4	6	3	5	1
röðun			8	7	5	4	6	3	5	1
8	1		7	6	5	4	1	3	5	8
7	1		6	5	5	4	1	3	7	8
6	1		5	4	5	3	1	6	7	8
5	1		5	4	1	3	5	6	7	8
4	1		4	3	1	5	5	6	7	8
3	1		3	1	4	5	5	6	7	8
2	1		1	3	4	5	5	6	7	8
raðað			1	3	4	5	5	6	7	8

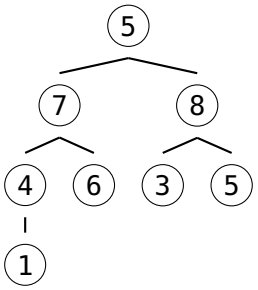
TAFLA 2: Rakning á Heapsort

Á næstu síðum eru myndir af breytingum á hrúgunni. MYND 1 sýnir fylkið eftir röðun með Heapsort. Hægt er að labba í gegnum fylkið með hefðbundnu aðferðinni (vísistök `1,2,...`).

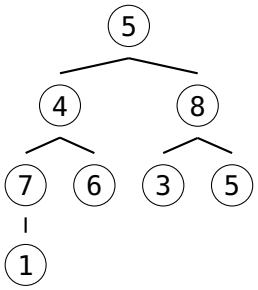
Þrep 1 — framkalla hrúguskilyrði



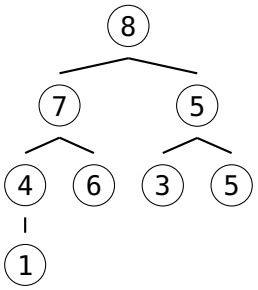
`sink(4,8)`



`sink(2,8)`

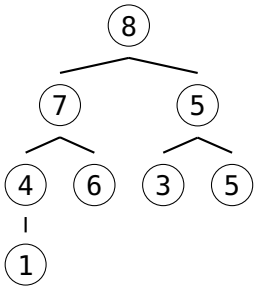


`sink(3,8)`

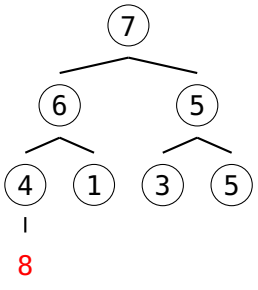


`sink(1,8)`

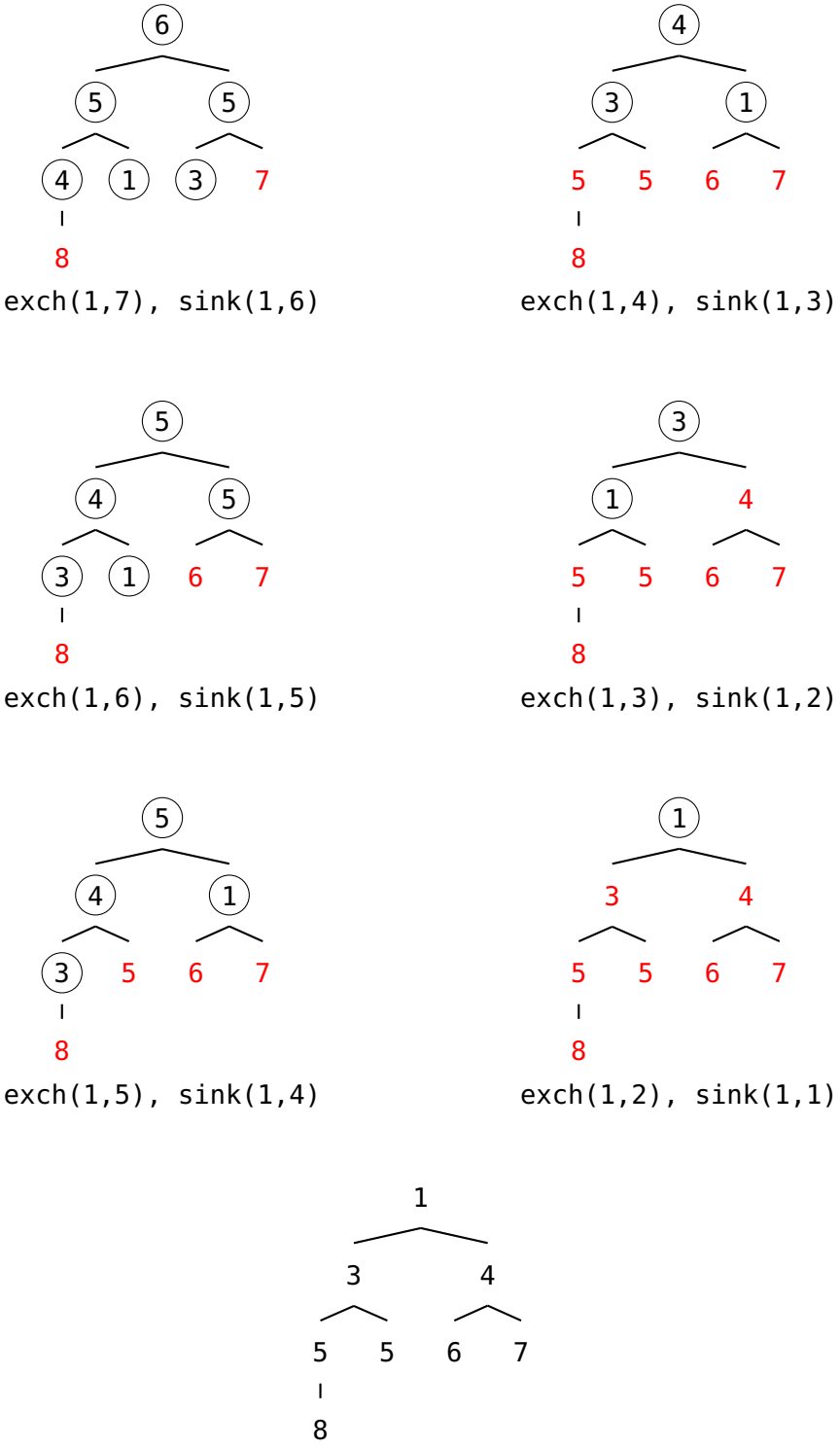
Þrep 2 — raða hrúgunni



`exch(1,8), sink(1,7)`



`exch(1,8), sink(1,7)`



MYND 1: Fullröðuð hrúga eftir Heapsort