Innlevering 1

Høst 2023

Teque

> Oppgaven er hentet fra Kattis[1]. Vi følger samme format på input- og output, slik at oppgaven deres kan lastes opp på Kattis, men dette er *ikke* et krav. Det er heller ikke nødvendig å oppfylle tidskravet som Kattis stiller.

Deque, eller *double-ended queue*, er en datastruktur som støtter effektiv innsetting på starten og slutten av en kø-struktur. Den kan også støtte effektivt oppslag på indekser med en array-basert implementasjon.

Dere skal utvide idéen om deque til *teque*, eller *triple-ended queue*, som i tillegg støtter effektiv innsetting i midten. Altså skal *teque* støtte følgende operasjoner:

`push_back(`x`)` sett elementet x inn bakerst i køen.

`push_front(`x`)` sett elementet x inn fremst i køen.

`push_middle(`x`)` sett elementet x inn i midten av køen. Det nylig insatte elementet x blir nå det nye midtelementet av køen. Hvis k er størrelsen på køen før innsetting, blir x satt inn på posisjon $\lfloor (k+1)/2 \rfloor$.

`get(`i`)` printer det i-te elementet i køen.

Merk at vi bruker 0-baserte indekser.

Input

Første linje av input består av et heltall N, der $1 \leq N \leq 10^6$, som angir hvor mange operasjoner som skal gjøres på køen.

Hver av de neste N linjene består av en streng S, etterfulgt av et heltall. Hvis S er `push_back`, `push_front` eller `push_middle`, så er S etterfulgt av et heltall x, slik at $1 \leq x \leq 10^9$. Hvis S er `get`, så S etterfult av et heltall i, slik at $0 \leq i <$ (størrelsen på køen).

Merk at du ikke trenger å ta høyde for ugyldig input på noen som helst måte, og du kan trygt anta at ingen `get`-operasjoner vil be om en indeks som overstiger størrelsen på køen.

[1] https://open.kattis.com/problems/teque

Output

For hver `get`-operasjon, print verdien som ligger på den i-te indeksen av køen.

Eksempel-input	Eksempel-output
9	3
`push_back` 9	5
`push_front` 3	9
`push_middle` 5	5
`get` 0	1
`get` 1	
`get` 2	
`push_middle` 1	
`get` 1	
`get` 2	

Oppgaver

(a) Skriv *pseudokode* for hver av operasjonene

- `push_back`
- `push_front`
- `push_middle`
- `get`

(b) Skriv et Java eller Python-program som leser input fra `stdin` og printer output *nøyaktig* slik som beskrevet ovenfor. Vi stiller ingen strenge krav til kjøretid.

(c) Oppgi en verste-tilfelle kjøretidsanalyse av samtlige operasjoner med $\mathcal{O}$-notasjon. I analysen fjerner vi begrensningen på N, altså kan N være vilkårlig stor.

(d) Hvis vi vet at N er begrenset, hvordan påvirker det kompleksiteten i $\mathcal{O}$-notasjon? Formulert annerledes: Hvorfor er det viktig at vi fjerner begrensningen på N i forrige deloppgave? (Hint: 10^6 er en konstant).

Sortering

I denne oppgaven skal du implementere flere sorteringsalgoritmer!

Disse to sorteringsalgoritmene *må* implementeres:

- Insertion sort
- Merge sort

Du kan implementere flere sorteringsalgoritmer dersom du ønsker.

For den ekstra ivrige: Du kan også implementere sorteringsalgoritmer som ikke er blitt gått gjennom ennå, eller som er utenfor pensum. Dersom du velger sorteringsalgoritmer som er utenfor pensum må du selv sørge for at implementasjonen er forståelig for retteren, og kan ikke forvente hjelp fra gruppelærer i like stor grad.

For alle deloppgavene er input det samme. Det som skiller deloppgavene er output.

Input

Det består av n heltall. For hvert tall i er det slik at $0 \leq i \leq 2^{31} - 1$. Input skal leses fra fil, hvor filnavnet blir gitt som kommandolinjeargument.

Eksempelinput:

```
80
91
7
33
50
70
13
321
12
```

Korrekthet

For små input-filer skal programmet ditt skrive en fil for hver sorteringsalgoritme du implementerer. For større inputfiler kan det være lurt å kun kjøre de raskeste algoritmene. Hver av outputfilene skal inneholde resultatet av sorteringen på samme format som input. Fordi alle algoritmene skal oppnå samme resultat, er det forventet at alle outputfilene er like.

Navngi hver outputfil etter navnet på inputfilen, etterfulgt av sorteringsalgoritmen som ble brukt til å sortere input. Hvis inputfilen heter `example` skal outputfilen hete `example_alg.out`, der `alg` byttes ut med navnet til den aktuelle sorteringsalgoritmen. Algoritmenavnet skal bare inneholde små bokstaver (`a`–`z`). For eksempel kan outputfilen for insertion sort få navnet `example_insertion.out`.

Du skal redgjøre for hvilke sorteringsalgoritmer som er implementert, og hva som er testet for å sjekke at alle algoritmene gir rimelige svar.

Gitt eksempelinputet ovenfor bør vi få følgende output.

Eksempeloutput:

```
7
12
13
33
50
70
80
91
321
```

Sammenligninger, bytter og tid

Hver av sorteringsalgoritmene skal kunne telle *antall sammenligninger* og *antall bytter* som gjøres i løpet av en sortering. Det kan være lurt å skrive egne metoder for sammenligning og bytter, slik at det blir enkelt å telle. I tillegg skal den totale tiden brukt på sortering oppgis i *mikrosekunder*. I Java kan tiden på din sorteringsalgoritme (i mikrosekunder) måles slik:

```
long t = System.nanoTime();

// din sorteringsalgoritme kjøres her

long time = (System.nanoTime()-t)/1000;
```

Tilsvarende for Python:

```
import time

t = time.time_ns()

// din sorteringsalgoritme kjøres her

timeus = (time.time_ns() - t) / 1000
```

Du skal kun produsere én outputfil for en gitt inputfil, som en CSV-fil. Dersom inputfilen heter `example`, skal outputfilen hete `example_results.csv`.

Hvis inputfilen inneholder n heltall, skal du gjøre $n+1$ sorteringer for hver sorteringsalgoritme du tester. Første sortering gjøres på det tomme arrayet, andre sortering gjøres på et array som inneholder det første elementet, tredje sortering gjøres på et array som inneholder de to første elementene, og så videre, helt opp til n.

Den første linjen i output skal se slik ut:

```
n, alg1_cmp, alg1_swaps, alg1_time, alg2_cmp, alg2_swaps, alg2_time
```

Det samme mønsteret skal følges for alle algoritmene du tester. De etterfølgende linjene skal bestå av kommaseparerte heltall.

- Første tall skal angi hvor mange elementer som ble sortert
- Andre tall skal angi hvor mange sammenligninger `alg1` brukte
- Tredje tall skal angi hvor mange bytter `alg1` gjorde
- Fjerde tall skal angi hvor lang tid `alg1` brukte (i mikrosekunder)
- Femte tall skal angi hvor mange sammenligninger `alg2` brukte
- ...

Eksempeloutput:

```
  n, insert_cmp, insert_swaps, insert_time, merge_cmp, merge_swaps, merge_time
  0,          0,            0,           1,         0,           0,          0
  1,          0,            0,           0,         0,           0,          1
  2,          1,            0,           2,         5,           2,          5
  3,          3,            2,           4,        10,           2,          6
  4,          6,            4,           6,        21,           5,         11
  5,          9,            6,           7,        28,           8,         15
  6,         12,            8,          10,        29,           9,         16
  7,         18,           13,          15,        40,          11,         21
  8,         19,           13,          15,        44,          12,         23
  9,         27,           20,          22,        45,          13,         23
```

Merk at mellomrom ignoreres.

Det er ikke forventet at deres algoritmer viser nøyaktig samme tall. Tallene er både avhengig av implementasjonen av sorteringsalgoritmen, og hvordan bytter og sammenligninger måles.

Det skal skrives kort om hvordan bytter og sammenligninger er målt.

Eksperimentér

Her skal du kun bruke programmet fra forrige deloppgave for å gjøre noen refleksjoner. Vi vil laste opp flere inputfiler på semestersiden som du kan bruke til å kjøre eksperimenter. Du bør kunne kjøre eksperimenter på ulike kombinasjoner av sorteringsalgoritmer (du trenger ikke lage et brukergrensesnitt for dette, men kan kommentere ut/inn kode utifra hva du trenger for å kjøre eksperimentene dine). Det er for eksempel lurt å kjøre eksperimenter hvor kun $\mathcal{O}(n \log(n))$ algoritmene kjøres, fordi da vil du kunne teste større inputfiler.

Filene som genereres er CSV-filer, og de kan åpnes i vanlige regnearkverktøy, som Excel, Numbers, etc. Her kan du lage enkle linjediagrammer, som kan gi deg innsikt i hvordan antall sammenligninger, bytter og tid henger sammen, samt hvor effektive de forskjellige sorteringsalgoritmene er.

Vi ønsker at du gjør noen egne refleksjoner. Du må ta stilling til disse spørsmålene:

- I hvilken grad stemmer kjøretiden overens med kjøretidsanalysene (store $\mathcal{O}$) for de ulike algoritmene?
- Hvordan er antall sammenligninger og antall bytter korrelert med kjøretiden?
- Hvilke sorteringsalgoritmer utmerker seg positivt når n er veldig liten? Og når n er veldig stor?
- Hvilke sorteringsalgoritmer utmerker seg positivt for de ulike inputfilene?
- Har du noen overraskende funn å rapportere?

Alle svar må begrunnes. Legg gjerne ved skjermbilder eller lignende som gir grunnlag for påstandendene dine. Vær tydelig på hvordan eksperimentet ble utført, slik at det lar seg reprodusere av andre.