# Høgskolen i Gjøvik

**Avdeling for elektro- og allmennfag**

# K O N T I N U A S J O N S E K S A M E N

| | | |
|---|---|---|
| **FAGNAVN:** | **Algoritmiske metoder** | |
| **FAGNUMMER:** | **LO 164A** | |
| **EKSAMENSDATO:** | **11. august 1995** | **TID: 09.00-14.00** |
| **FAGLÆRER:** | **Frode Haug** | **KLASSE: 2 AA / AE** |
| **ANTALL SIDER UTLEVERT:** | **5 (inkludert denne forside)** | |
| **TILLATTE HJELPEMIDLER:** | **Alle trykte og skrevne** | |

**NB: - Kontroller at alle oppgavearkene er tilstede.**

**- LES HELE OPPGAVETEKSTEN NØYE, FØR DU BEGYNNER Å BESVARE NOE SOM HELST.**

**- DET ER INGEN SAMMENHENG MELLOM DE ULIKE DELENE I OPPGAVENE 2 OG 3. DERMED KAN ALLE UNDERPUNKTER LØSES TOTALT UAVHENGIG.**

**- Ikke skriv noe av din besvarelse på oppgavearkene.**

**- Kladd og oppgavearkene leveres sammen med besvarelsen. Kladd merkes med "KLADD".**

**- Husk kandidatnummer på alle ark (IKKE oppgavearkene).**

# Oppgave 1 (teori, 5 %)

Av de sorteringsmetodene som vi har lært om, så er Shellsort, Quicksort og Mergesort generelt de raskeste.

**a)** Forklar kort og klart, med egne ord, hvordan hver av disse fungerer.

**b)** Hvilke karakteristiske egenskaper/fordeler har hver av dem ?

# Oppgave 2 (teori, 20 %)

I hele oppgave 2 er det key'ene "K O N T I N U A S J O N S E K S A M E N" (i denne rekkefølge fra venstre til høyre, og blanke regnes ikke med) som du skal bruke. For alle oppgavene så gjelder det at den initielle heap, tre eller array er tom før første innlegging ("Insert") utføres.

Tegn (skriv) den resulterende datastruktur når key'ene legges inn i:

**a)** et binært søketre.

**b)** en heap.

**c)** et 2-3-4 tre.

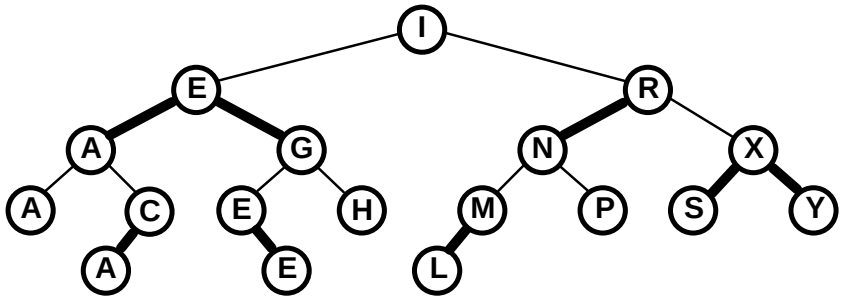
**d)** et Red-Black tre.

**e)** en array vha. linear probing. Arrayen har indeksene 0-19. Bruk hash-funksjonen: $h_1(k) = k \bmod 20$, der "k" står for bokstavens nummer i alfabetet (1-29).

# Oppgave 3 (teori, 20 %)

Denne oppgaven inneholder tre totalt uavhengige oppgaver, som er basert på ulike deler av pensum.

**a)** Vi har et Red-Black tre med følgende utseende (jfr. fig.15.8 s.222 i læreboka.) :



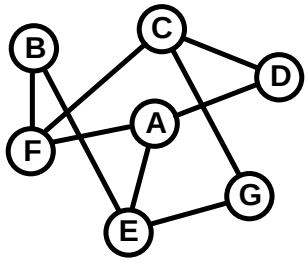

- Hvilke linker oppdateres og
- hvilke "farger" (Red/Black) får de

av "Split"- og "Rotate"-funksjonene (s.225 og 226 i læreboka) når bokstaven "R" og deretter "F" blir lagt inn i dette treet ? (Tegn opp.)

**b)** Vi har søkestrengen "APRIKOSEKOMPOTTKOMPRESJON". Mønsteret vi skal søke med er "KOMPRE". Tegnene som totalt kan forekomme i søkestrengen er bokstavene A-Å.

- Angi skip-arrayen for dette tilfellet.
- Tegn opp hvordan mønsteret forflyttes bortover i søkestrengen når søkealgoritmen s.288 i læreboka utføres. Dvs. lag en figur etter samme prinsipp som fig.19.7 s.287 i læreboka.

**c)** Vi har grafen:




**c1)** Skriv opp nabomatrisen for dette tilfellet.

**c2)** Tegn dybde-først søketreet for dette tilfellet (dvs. ved bruk av nabomatrise), når "Search" (s.424) og "Visit" (s.427) fungerer som angitt i læreboka.

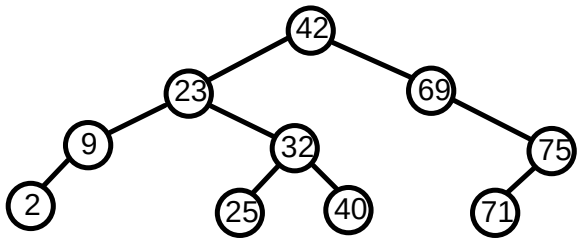
# Oppgave 4 (teori og koding, 5 + 5 + 20 %)

I denne oppgaven skal du bygge opp et binært tre av noder. Disse nodene inneholder kun et heltall og pekere til henholdsvis venstre og høyre subtre. Treet lages/genereres etterhvert ved at du leser tre og tre sammenhørende tall fra en fil (til filen er tom). Det første tallet angir heltallet i noden. Det 2. og 3. tallet er enten 0 eller 1. Disse angir om noden har henholdsvis et venstre og et høyre subtre (0 = har ikke subtre, 1 = har subtre på denne siden). Data om nodene kommer i **PREFIKS ORDEN** på filen. Om filen er tom, betyr det at treet også er tomt.

**a) (5 %)** Tegn treet, dersom filen har følgende innhold:

```
 7  1  1
 4  1  1
 2  1  1
 1  0  0
 3  0  0
 5  0  1
 6  0  0
10  1  1
 8  0  1
 9  0  0
13  1  1
12  1  0
11  0  0
14  0  0
```

**b) (5 %)** Skriv filens innhold, dersom treet har følgende utseende:

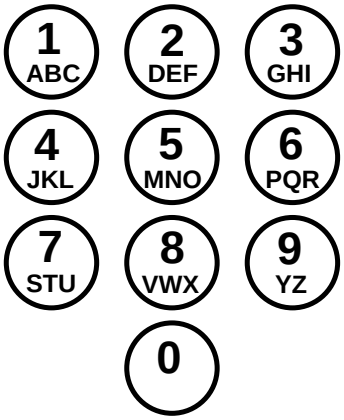



**c) (20 %)** Skriv kode med:

- deklarasjon av noden.
- definisjon av pekeren til roten for hele treet.
- "main" med kallet som starter det hele.
- en **rekursiv funksjon** som sørger for at treet leses inn og genereres, dvs:
  - leser inn data om en ny node (alt på en linje i filen).
  - returnerer en peker til noden.
  - nodens subtrær påhektes (ved rekursive kall til seg selv)
  - kaller funksjonen "feil" om det ikke kommer det rette antall noder/linjer.

# Oppgave 5 (koding, 25 %)

Deler av tastaturet på dagens (mobil)telefoner kan f.eks. se ut som:




Dvs. hvert siffer kan forbindes med opptil tre bokstaver. Unntakene er sifrene '0' og '9'.
Dette medfører at telefonnummeret "35726527" bl.a. kan forbindes med teksten "INTERNET", men nummeret kan også forbindes med den mere meningsløse teksten "GMUFRODS".

Din oppgave blir å generere alle mulige bokstavsekvenser som kan oppstå på grunnlag av et gitt 8-sifret telefonnummer.

Dette skal løses ved hjelp av de globale variablene:

- char bokstav [10] [4] = { " ", "ABC", "DEF", "GHI", "JKL", "MNO", "PQR", "STU", "VWX", "YZ "};
  Fire lang, da må ha plass til '\0' (tillegges automatisk).
  **NB: Legg merke til innholdet i element nr. 0 og 9: en blank for '0' og en blank til slutt i "YZ ".**
- char nummer[9]; 8-sifret tlf.nr. med indeks 0-7, pluss '\0' i den 8.posisjonen.
- char tekst[9]; Teksten som til enhver tid er generert av "lag_tekst".

"Main", som starter det hele, ser ut som:

```
int main()  {
  cout << "8-sifret telefonnummer:  ";   cin.get(nummer, 9);
  lag_tekst(0);                          //  Starter med sifferet med indeks nr.0.
  return 0;
}
```

**Du skal skrive en rekursiv funksjon ("lag_tekst") som genererer og skriver ut alle mulige 8-bokstavsekvenser (også inneholdende blanke), vha. de globale variablene ovenfor. Duplikater skal kun tas vekk for sifferet '0'.**
**Hvor mange løsninger finnes maksimalt for et 8-sifret nummer?**