Frågor och svar från föreläsning 3: Rekursion, callstack och minne

Frågor:

1) Vad är rekursion?

2) Vad innebär divide-and-Conquer?

3) Hur fungerar fibonaccisekvensen? (Själva talsekvensen, inte någon algoritmisk implementation för att beräkna fibonaccinummer!)

4) Hur fungerar stackminnet i Java?

5) Varför är en rekursiv fibonaccialgoritm så beräkningstung? Vad har den för tidskomplexitet?

6) När man anropar en rekursiv fibonaccimetod med höga nummer kommer resultatet plötsligt att börja bli negativt. Varför?

7) Vad är tidskomplexiteten för en binär sökalgoritm? Varför?

8) Vad är skillnaden mellan O(n*logn), O(logn) och O(n)?

9) Vad är en naiv algoritm för någonting?

10) När vi deklarerar en int, var någonstans i minnet sparas den?

11) Beskriv vad som händer när vi instansierar ett objekt och sen skickar in det som argument i en metod. Var sparas objektet? Vad skickas till metoden?

12) Vad är dynamisk programming för någonting? Varför är det bra att känna till?

13) Vad menas med LIFO? Vilken datastruktur är det förknippat med?

14) Vad är skillnaden mellan att skicka någonting som värde och att skicka någonting som referens?

15) Vad innebär primitiv rekursion?

16) Vad är kontrollflöden?

Facit:

1) Rekursion inom programmering är en metod som anropar sig själv. Mer formellt är det någonting som definieras i termer av sig självt.

2) Divide-and-conquer innebär att algoritmen halverar problemet för varje gång den körs. En binär sökalgoritm är ett exempel på divide-and-conquer och har tidskomplexiteten O(logn).

3) I fibonaccisekvensen är varje nytt tal summan av de två föregående. Formeln är: Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2), där n är det tal i sekvensen som vi vill beräkna.

4) Stacken är det minne i programmet där primitiva datatyper och activation frames för metodanrop sparas. Den fungerar som ungefär som en stapel: man lägger på saker och när man vill ta bort dem igen måste man börja med det man sist la på. Det är därför man säger att den fungerar enligt minnesregeln LIFO: Last In, First Out.

5) Den är beräkningstung eftersom varje nytt rekursivt anrop producerar två nya anrop i gengäld. Varje gång n ökar med 1 fördubblas tiden. Man säger att den har exponentiell tillväxt, med tidskomplexiteten O(2n).

6) Programmet kommer att börja skriva ut negativa värden när fibonaccialgoritmen anropas med tal större än 161 eftersom storleken på dessa tal kommer att överskrida kapaciteten för vad man kan lagra i en long, som är den största primitiva datatypen i Java.

7) Varje gång en binär sökfunktion anropar sig själv halveras datamängden eftersom vi kan utesluta halva listan (vi vet ju om talet är större eller mindre än mittvärdet).
Tidskomplexiteten för binär sökalgoritm är alltså O(logn) eftersom den halverar datamängden för varje nytt metodanrop.

8)
O(logn) : Mängden n halveras för varje anrop.
O(n): Tiden ökar linjärt: när antalet n dubblas dubblas också körtiden.
O(n*logn) : Tiden ökar linjärt med antalet n, multiplicerat med logn.

9) En naiv algoritm är den enklaste lösningen på ett problem, men väldigt sällan den bästa.

10) En int sparas på stacken i Java.

11) När vi instansierar ett objekt sparas det i heapminnet. En referens till objektet sparas på stacken, och det är referensen som skickas med som argument i parametrarna till en metod. Det är därför vi t.ex. kan skapa en void-metod som tar in en array och sorterar den utan att

sedan returnera en array igen: både metoden och kodblocket där arrayen deklarerades håller samma referens.

12) Dynamisk programmering är en optimeringsteknik där man kan använda antingen en top-down-approach (memoisering) eller en bottom-up-approach (tabulering) för att optimisera rekursiva och iterativa algoritmer.

13) LIFO står för Last In, First Out och beskriver hur en stack fungerar.
14) När vi skickar någonting som värde till en metod kopieras värdet. Det innebär att om du skickar en int-variabel till en metod och ändrar värdet på den inuti metoden så kommer inte ursprungsvariabeln att ändra värde. Alla primitiva datatyper skickas automatiskt som värde i Java.

15) Primitiv rekursion innebär att det går att skriva om algoritmen så att rekursiva anropen ersätts med for-loopar i stället.

16) Kontrollflöden är strukturer som manipulerar programflödet på något vis. Exempelvis if-satser (selektion), loopar (iteration) och självanropande metoder (rekursion).