Universität Innsbruck - Institut für Informatik
Prof. Clemens Ballarin, Robert Binna, Friedrich Neurauter, Francois Scharffe and Sarah Winkler

29. April 2008

Proseminar Algorithmen und Datenstrukturen

Exercise sheet 6

Exercise 1 (O Notation)

Compare the following pairs of functions in terms of order of magnitude. In each case, say whether $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$ and/or $f(n) \in \Theta(g(n))$.

	$f(n)$	$g(n)$
a)	$n^3 + 3n + 1$	n^4
b)	$\log(n)$	n
c)	$n \cdot 2^n$	3^n
d)	n	$(\log(n))^5$
e)	$\log(n)$	$\log(n^2)$
f)	$100n + \log(n)$	$n + (\log(n))^2$

In order to show that $f(n) \in O(g(n))$ we have to find two constants c and N such that for all $n > N$ we have that $f(n) \leq c \cdot g(n)$. Formally,

$$f(n) \in O(g(n)) \Longleftrightarrow \exists c > 0 \ \exists N \ \forall n > N \quad f(n) \leq c \cdot g(n) \tag{1}$$

Moreover, $f(n) \in \Omega(g(n))$ if and only if $g(n) \in O(f(n))$, and $f(n) \in \Theta(g(n))$ if and only if $f(n) \in O(g(n))$ and $g(n) \in O(f(n))$.

a) n^3+3n+1 is in $O(n^4)$. Setting $c = 1$ and $N = 1$, we have to show that $n^3+3n+1 \leq n^4$ for all $n \geq 2$. A simple proof by induction on n will suffice.

$\boxed{n = 2}$ $2^3 + 3 * 2 + 1 = 15 \leq 16 = 2^4$

$\boxed{n > 2}$ The induction hypothesis (IH) is: $n^3 + 3n + 1 \leq n^4$, and we must show that $(n+1)^3 + 3(n+1) + 1 \leq (n+1)^4$.
$(n+1)^3+3(n+1)+1 = n^3+3n^2+3n+1+3n+3+1 = (n^3+3n+1)+3n^2+3n+4$. By the IH we get that $(n^3 + 3n + 1) + 3n^2 + 3n + 4 \leq n^4 + 3n^2 + 3n + 4$, and it remains to show that this is smaller than or equal to $(n+1)^4$. But this is easy:

$$\begin{aligned} n^4 + 3n^2 + 3n + 4 &\leq (n+1)^4 \\ n^4 + 3n^2 + 3n + 4 &\leq n^4 + 4n^3 + 6n^2 + 4n + 1 \\ 3 &\leq 4n^3 + 3n^2 + n \end{aligned}$$

And this last inequality is trivially satisfied for $n > 2$.

Alternatively, we can use the following reasoning to establish that $n^3 + 3n + 1$ is in $O(n^4)$.

$$\begin{aligned} n^3 + 3n + 1 &\leq 3n^3 + 3n + 3 \quad \text{for } n \geq 1 \\ &\leq 3n^3 + 3n^3 + 3n^3 \\ &\leq 9n^3 \\ &\leq 9n^4 \end{aligned}$$

Note that this proof also yields that $n^3 + 3n + 1$ is in $O(n^3)$. Finally, for the ones who are familiar with computing with limits, $\lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ (i.e., the limit exists) implies condition (1). Hence,

$$\lim_{n\to\infty} \frac{n^3 + 3n + 1}{n^4} = \lim_{n\to\infty} \frac{1}{n} + 3\frac{1}{n^3} + \frac{1}{n^4} = 0$$

also tells us that $n^3 + 3n + 1$ is in $O(n^4)$.

Clearly, n^4 is not in $O(n^3+3n+1)$. The proof goes by contradiction. So assume that there exist constants c and N such that for all $n > N$ we have $n^4 \leq c(n^3+3n+1)$. This means that

$$c \geq \frac{n^4}{n^3 + 3n + 1} \geq \frac{n^4}{3n^3 + 3n + 3} \geq \frac{n^4}{9n^3} = \frac{n}{9}$$

Hence, $c \geq \frac{n}{9}$ for all $n > N$. Obviously such a constant c does not exist.

As a consequence of the results obtained above, $n^3 + 3n + 1$ is not in $\Theta(n^4)$.

b) For $n > 0$ we have $\log(n) \leq n$ if and only if $e^{\log(n)} \leq e^n$, which simplifies to $n \leq e^n$ (remember that the exponential function is monotonically increasing). Establishing $n \leq e^n$ for $n > 0$ is a trivial task if we consider the power series representation of the exponential function:

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \ldots$$

Hence, $\log(n)$ is in $O(n)$. Reasoning with limits we obtain the same result somewhat faster

$$\lim_{n\to\infty}\frac{\log(n)}{n} = \lim_{n\to\infty}\frac{1/n}{1} = \lim_{n\to\infty}\frac{1}{n} = 0$$

by an application of de L'Hopital's rule for calculating limits.

However, n is not in $O(\log(n))$. Again the proof goes by contradiction. So assume that there exist constants c and N such that for all $n > N$ we have $n \leq c \cdot \log(n)$. This means that $c \geq n/\log(n)$, which implies that

$$e^c \geq e^{\frac{n}{\log(n)}} = e^{\frac{n\log(n)}{(\log(n))^2}} = n^{\frac{n}{(\log(n))^2}} \geq n$$

(note that $n \geq (\log(n))^2$ for $n \geq 1$, a fact which we will prove later). Hence, $e^c \geq n$ for all $n > N$. Obviously such a constant c does not exist.

As a consequence of the results obtained above, $\log(n)$ is not in $\Theta(n)$.

c)

$$\lim_{n\to\infty}\frac{n\cdot 2^n}{3^n} = \lim_{n\to\infty}\frac{n}{(\frac{3}{2})^n} = \lim_{n\to\infty}\frac{1}{(\frac{3}{2})^n\cdot\log(\frac{3}{2})} = 0$$

Hence, $n \cdot 2^n$ is in $O(3^n)$.

However, 3^n is not in $O(n \cdot 2^n)$ as a simple proof by contradiction shows.

As a consequence of the results obtained above, $n \cdot 2^n$ is not in $\Theta(3^n)$.

d) $\log(n))^5$ is in $O(n)$. It is even true that $(\log(n))^r$ is in $O(n)$ for every natural number $r \geq 1$.

Proof. So we have to find constants $c > 0$ and N such that $(\log(n))^r \leq c \cdot n$ for all $n > N$. The logarithm of n is non-negative for $n \geq 1$ and therefore $(\log(n))^r \leq c\cdot n$ is equivalent to $\log(n) \leq \sqrt[r]{c\cdot n}$. Now we conclude

$$\log(n) \leq \sqrt[r]{c\cdot n} \iff e^{\log(n)} \leq e^{\sqrt[r]{c\cdot n}} \iff n \leq e^{\sqrt[r]{c\cdot n}} \iff \frac{e^{\sqrt[r]{c\cdot n}}}{n} \geq 1$$

Next we use the power series representation of the exponential function to simplify $\frac{e^{\sqrt[r]{c\cdot n}}}{n}$.

$$\frac{e^{\sqrt[r]{c\cdot n}}}{n} = \frac{\sum_{k=0}^{\infty}\frac{(\sqrt[r]{c\cdot n})^k}{k!}}{n} = \frac{\sum_{k=0}^{\infty}\frac{(c\cdot n)^{\frac{k}{r}}}{k!}}{n} = \sum_{k=0}^{\infty}\frac{c^{\frac{k}{r}}n^{\frac{k}{r}-1}}{k!}$$

Remember that we want to find constants $c > 0$ and N such that this sum becomes 1 or greater for all $n > N$. Analyzing the structure of the sum, we see that all summands are non-negative. Hence, if only one of the summands is equal to 1 or greater then the whole sum is, too. Moreover, considering the summand for $k = r$

$$\frac{c^{\frac{r}{r}}n^{\frac{r}{r}-1}}{r!} = \frac{cn^0}{r!} = \frac{c}{r!}$$

we see that it is independent of n (which is very good!). So, if we set c to the faculty of r then the entire sum becomes 1 or greater, and $(\log(n))^r \leq r! \cdot n$ for all $n \geq 1$. □

However, n is not in $O((\log(n))^r)$ as a proof by contradiction similar to the one used in b) shows.

As a consequence of the results obtained above, n is not in $\Theta((\log(n))^r)$.

e) Note that $\log(n^2) = 2 \cdot \log(n)$. Hence, $\log(n)$ is in $O(\log(n^2))$ and $\log(n^2)$ in $O(\log(n))$, which implies that $\log(n)$ is in $\Theta(\log(n^2))$.

f)

$$\lim_{n\to\infty} \frac{100n + \log(n)}{n + (\log(n))^2} = \lim_{n\to\infty} \frac{100 + \frac{1}{n}}{1 + 2(\log(n))\frac{1}{n}} = \lim_{n\to\infty} \frac{100n + 1}{n + 2(\log(n))} = \lim_{n\to\infty} \frac{100}{1 + 2\frac{1}{n}} = 100$$

Hence, $100n + \log(n)$ is in $O(n + (\log(n))^2)$. The same calculation (with nominator and denominator exchanged) tells us that $n + (\log(n))^2$ is in $O(100n + \log(n))$. Together this implies that $100n + \log(n)$ is in $\Theta(n + (\log(n))^2)$.

Exercise 2 (O Notation)

The addition theorem for O states that $f(n) \in O(s(n))$ and $g(n) \in O(r(n))$ imply that $f(n) + g(n) \in O(s(n) + r(n))$. Now formulate the corresponding theorem for Ω and provide a proof for it.

Claim: $f(n) \in \Omega(s(n))$ and $g(n) \in \Omega(r(n))$ imply that $f(n) + g(n) \in \Omega(s(n) + r(n))$.

Proof. Let's assume that $f(n) \in \Omega(s(n))$ and $g(n) \in \Omega(r(n))$. By definition of Ω, we have

$$\exists c_1 > 0\ \exists N_1\ \forall n > N_1 \quad f(n) \geq c_1 \cdot s(n)$$
$$\exists c_2 > 0\ \exists N_2\ \forall n > N_2 \quad g(n) \geq c_2 \cdot r(n)$$

Hence, $f(n) + g(n) \geq c_1 \cdot s(n) + c_2 \cdot r(n) \geq \min(c1, c2) \cdot (s(n) + r(n))$. Now we can prove that $\exists c > 0\ \exists N\ \forall n > N \quad f(n) + g(n) \geq c \cdot (s(n) + r(n))$ by setting N to $\max(N_1, N_2)$ and c to $\min(c_1, c_2)$. □

Exercise 3 (O Notation)

a) Find a counterexample to the following claim: $f(n) \in O(s(n))$ and $g(n) \in O(r(n))$ imply that $f(n) - g(n) \in O(s(n) - r(n))$.

b) Find a counterexample to the following claim: $f(n) \in O(s(n))$ and $g(n) \in O(r(n))$ imply that $f(n)/g(n) \in O(s(n)/r(n))$.

a) Let $f(n) = 2n \in O(n)$ and $g(n) = n \in O(n)$.

b) Let $f(n) = n \in O(n)$ and $g(n) = 1/n \in O(1)$.

Exercise 4 (O Notation)

Prove or disprove (by means of a counterexample) the following statements:

a) $O(f(n) + g(n)) = O(\max(f(n), g(n)))$

b) $O(f(n) + g(n)) = O(\min(f(n), g(n)))$

c) $\Omega(f(n) + g(n)) = \Omega(\max(f(n), g(n)))$

d) $\Omega(f(n) + g(n)) = \Omega(\min(f(n), g(n)))$

e) $\Theta(f(n) + g(n)) = \Theta(\max(f(n), g(n)))$

f) $\Theta(f(n) + g(n)) = \Theta(\min(f(n), g(n)))$

First, we prove a simple theorem concerning O notation. For every constant $k > 0$ and $f : \mathbb{N}_0 \to \mathbb{R}_0^+$, we have

$$O(k \cdot f(n)) = O(f(n))$$

Proof. For all functions $g : \mathbb{N}_0 \to \mathbb{R}_0^+$ we have

$$\begin{aligned}
g \in O(k \cdot f(n)) \iff & \exists c > 0\ \exists N\ \forall n > N \quad g(n) \leq c \cdot (k \cdot f(n)) \\
\iff & \exists c > 0\ \exists N\ \forall n > N \quad g(n) \leq (c \cdot k) \cdot f(n) \\
\iff & \exists c' > 0\ \exists N\ \forall n > N \quad g(n) \leq c' \cdot f(n) \\
& g \in O(f(n))
\end{aligned}$$

$\square$

a) $O(f(n) + g(n)) = O(\max(f(n), g(n)))$ is true. If f and g are functions from $\mathbb{N}_0$ to $\mathbb{R}_0^+$ then the following inequalities hold for all n:

$$\max(f(n), g(n)) \leq f(n) + g(n) \leq 2 \cdot \max(f(n), g(n))$$

From the first inequality we conclude that $O(\max(f(n), g(n))) \subseteq O(f(n) + g(n))$, since any function that is bounded by $\max(f, g)$ is also bounded by $f + g$.

Likewise, we conclude from the second inequality that $O(f(n) + g(n)) \subseteq O(2 \cdot \max(f(n), g(n)))$. But by the above theorem $O(2 \cdot \max(f(n), g(n))) = O(\max(f(n), g(n)))$, and hence $O(f(n) + g(n)) \subseteq O(\max(f(n), g(n)))$.

b) $O(f(n) + g(n)) = O(\min(f(n), g(n)))$ is false. For example, let $f(n) = n$ and $g(n) = n^2$. Then $O(f(n) + g(n)) = O(n + n^2)$ which is equal to $O(n^2)$ by a), whereas $O(\min(f(n), g(n))) = O(n)$.

c) $\Omega(f(n)+g(n)) = \Omega(\max(f(n), g(n)))$ is true by similar reasoning as in a). If f and g are functions from $\mathbb{N}_0$ to $\mathbb{R}_0^+$ then the following inequalities hold for all n:

$$\max(f(n), g(n)) \leq f(n) + g(n) \leq 2 \cdot \max(f(n), g(n))$$

From the first inequality we conclude that $\Omega(f(n) + g(n)) \subseteq \Omega(\max(f(n), g(n)))$, since any function that is bounded from below by $f+g$ is also bounded by $\max(f, g)$.

Likewise, we conclude from the second inequality that $\Omega(2 \cdot \max(f(n), g(n))) \subseteq \Omega(f(n) + g(n))$. But $\Omega(2 \cdot \max(f(n), g(n))) = \Omega(\max(f(n), g(n)))$, and hence $\Omega(\max(f(n), g(n))) \subseteq \Omega(f(n) + g(n))$.

d) $\Omega(f(n) + g(n)) = \Omega(\min(f(n), g(n)))$ is false by similar reasoning as in b).

e) $\Theta(f(n) + g(n)) = \Theta(\max(f(n), g(n)))$ is true. By definition of Θ, we have $h \in \Theta(f(n) + g(n))$ if and only if $h \in O(f(n) + g(n))$ and $h \in \Omega(f(n) + g(n))$. According to a) and c) this is equivalent to $h \in O(\max(f(n), g(n)))$ and $h \in \Omega(\max(f(n), g(n)))$, which is equivalent to $h \in \Theta(\max(f(n), g(n)))$.

f) $\Theta(f(n) + g(n)) = \Theta(\min(f(n), g(n)))$ is false by similar reasoning as in b).

Exercise 5 (O Notation)

Assuming that the bodies of the for-loops only contain simple statements (no loops, no function calls etc.), and hence have constant time complexity, what is the time complexity of each of the following code fragments?

a)

```
for (i = 0; i < N; i++) {
  ...
}
for (j = 0; j < M; j++) {
  ...
}
```

b)

```
for (i = 0; i < N; i++) {
  for (j = 0; j < N; j++) {
    ...
  }
}
for (k = 0; k < N; k++) {
  ...
}
```

c)
```
for (i = 0; i < N; i++) {
  for (j = i; j < N; j++) {
    ...
  }
}
```

d) Consider this sample implementation for the longest upsequence algorithm (exercise 4 on exercise sheet 3). What is its complexity expressed in O notation? Consider both cases, binary and naive linear search.

```
#include <stdio.h>

int main() {
  int n = 8; /* array length */
  int X[] = {1, 3, 4, 6, 2, 4, 4, 0};
  int M[n]; for (int i=0; i<n; i++) M[i]=0;
  int i;

  int k = 0;
 M[0] = X[0];
  for(i = 1; i < n; i++){
    if (X[i] >= M[k]) {
      k++;
      M[k] = X[i];
      }
    else if (X[i] < M[0]) {
      M[0] = X[i];
      }
    else {
      /* find j such that M[j-1] <=X[i]<M[j] */
      /* Possibility 1: naive search
      int j = i;
      while (M[j-1] > X[i])
        j--; */
      /* Possibility 2: binary search */
      int b, j = 1;
      for (b = (k+1)/2; M[j-1] > X[i] || X[i] >= M[j]; b /= 2)
        if (M[j-1] > X[i])
          j -= b;
        else
          j += b;
      M[j] = X[i];
    }
  }
```

```
    printf("Longest upsequence has length %d\n", k + 1);
    return 0;
}
```

a) The first loop is $O(N)$ and the second loop is $O(M)$. Since we don't know which one is bigger, we say this is $O(N+M)$, or equivalently $O(\max(N,M))$. In the case where the second loop goes to N instead of M the complexity is $O(N)$. You can see this from either expression above. $O(N+M)$ becomes $O(2N)$, and when you drop the constant it is $O(N)$. $O(\max(N,M))$ becomes $O(\max(N,N))$ which is $O(N)$.

b) The first set of nested loops is $O(N^2)$ and the second loop is $O(N)$. Hence, the overall complexity is $O(\max(N^2,N))$, which is $O(N^2)$.

c) When i is 0, the inner loop executes N times. When i is 1, the inner loop executes $N-1$ times. In the last iteration of the outer loop, when i is $N-1$, the inner loop executes exactly once. Hence, the number of times the body of the inner loop executes is $N + (N-1) + ... + 2 + 1 = \sum_{i=1}^{N} i = N(N+1)/2$, which amounts to $O(N^2)$.

d) The for-loop executes $n-1$ times, where n is the length of the input sequence. Most of the body of the for-loop is made up of basic statements that execute in constant time (i.e., the execution time does not depend on n), only the part where we search for a position to place the current element depends on n. Hence, (in the worst case) the complexity of the body of the for-loop is the complexity of the search algorithm we use.

   In case of linear search, the iteration count of the while-loop is determined by the variable j which is initialized to the value of i, the loop index of the main loop. In the worst case j is $n-1$. Consequently, the complexity of the search is $O(n)$.

   In case of binary search, the iteration count of the for-loop is determined by the variable b, which is initialized to $(k+1)/2$. In the worst case b is $n/2$. Hence, the complexity of the search is $O(\log n)$.

   Thus the total complexity is $O(n^2)$ using linear search and $O(n \cdot \log n)$ in case of binary search.

Exercise 6 (Linked List)

Reconsider your linked list implementation of last week, try to eliminate the bugs it has (if it has any) and provide a demo program that is equipped with a user interface. The user interface should consist of a menu that has entries for

- adding elements to a list,
- removing elements from a list,
- printing a list,

- inserting elements in order and
- searching for a specific element.

The main program main.c:

```c
#include <stdio.h>
#include <string.h>
#include "list.h"

void printMenu()
{
  fprintf(stdout,"\n*** List Menu ************************\n");
  fprintf(stdout, "*  [a] Add an element to the list      *\n");
  fprintf(stdout, "*  [r] Remove an element from the list *\n");
  fprintf(stdout, "*  [p] Print the list                  *\n");
  fprintf(stdout, "*  [s] Search for an element           *\n");
  fprintf(stdout, "*  [q] Quit the program                *\n");
  fprintf(stdout, "****************************************\n");
  fprintf(stdout,"\n");
}

char chooseCmd()
{
  char c = '\0';
  char* s = NULL;
  do
  {
    fprintf(stdout, "Make your selection: ");
    c = getchar();
    //flush stdin: note fflush has undefined behaviour for input streams
    while (getchar() != '\n') ;
  } while ( !(s = strchr("arpsq",c)) );
  fprintf(stdout, "\n");
  return *s;
}

//read an integer from stdin, using the string s as prompt
int readInt(const char *s)
{
  int i = 0, ret = 0;

  do
  {
    if (s)
```

```
        printf("%s", s);
        ret = scanf("%d", &i);
        //flush stdin: note fflush has undefined behaviour for input streams
        while (getchar() != '\n') ;
    } while (ret != 1);

    return i;
}

int main()
{
    list_t* list = NULL; //empty list

    int stop = 0;;
    while (!stop)
    {
        printMenu();
        switch ((unsigned int) chooseCmd())
        {
            case 'a':
                list = addElement(list,
                        readInt("Enter an element to add to the list: ")
                );
                break;
            case 'r':
                list = removeElement(list,
                        readInt("Enter an element to remove from the list: ")
                );
                break;
            case 'p':
                printList(list);
                break;
            case 's':
                {
                    int res = searchList(list,
                        readInt("Enter an element to search for in the list: ")
                    );
                    printf("\nThis element is%s present in the list\n", res ? "" : " not"
                    break;
                }
            case 'q':
                stop = 1;
                break;
            default:
```

```
            break;
        }
    }
    //free the memory occupied by the list nodes
    deleteList(list);
    return 0;
}
```

The list interface file list.h:

```
#ifndef LIST_H
#define LIST_H

typedef struct list
{
    int data;
    struct list* next;
} list_t;

list_t* addElement(list_t* list, int x);

void printList(const list_t* list);

void deleteList(list_t* list);

int searchList(const list_t* list, int x);

list_t* removeHead(list_t* list);

list_t* removeElement(list_t* const list, int x);

#endif
```

The list implementation file list.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "list.h"

list_t* addElement(list_t* list, int x)
{
    list_t* p = (list_t*) malloc(sizeof(list_t));
    if (p)
    {
        p->data = x;
```

```
    p->next = list;
  }
  return p;
}

void printList(const list_t* list)
{
  fprintf(stdout, "[");
  while (list)
  {
    fprintf(stdout, "%d", list->data);
    if (list->next)
      fprintf(stdout, ";");
    list =  list->next;
  }
  fprintf(stdout, "]\n");
}

void deleteList(list_t* list)
{
  while (list)
  {
    list_t* p = list->next;
    free(list);
    list = p;
  }
}

int searchList(const list_t* list, int x)
{
  while (list)
  {
    if (list->data == x)
      return 1;
    list = list->next;
  }
  return 0;
}

list_t* removeHead(list_t* list)
{
  list_t* p = NULL;

  if (list)
```

```
  {
    p = list->next;
    free(list);
  }
  return p;
}

list_t* removeElement(list_t* const list, int x)
{
  if (!list)
    return NULL;
  if (list->data == x)
    return removeHead(list);

  list_t* prev = list;
  list_t* p = list->next;

  while (p)
  {
    if (p->data == x)
    {
      assert(prev != NULL);
      prev->next = p->next;
      free(p);
      return list;
    }
    else
    {
      prev = p;
      p = p->next;
    }
  }
  return list;
}
```