Algorithms and Data Structures I. Example test 1

Tibor Ásványi
Department of Computer Science
Eötvös Loránd University, Budapest
asvanyi@inf.elte.hu

May 6, 2023

In exercise 1 of Section 1 do not write code: Illustrate the algorithms as you saw them in the classroom. In exercises 2-5 of Section 1 and in Section 2 try to write efficient code (structure diagrams). Deletion includes explicit deallocation. Specify the types and modes of the formal parameters of the functions and procedures and the return types of functions.

In the test, there will be 3 exercises about illustrating the three sorting methods (3*10 points) and two assignments about drawing structure diagrams (2*15 points, one exercise on one-way lists and one about two-way lists).

1 Introductory exercises

1a Illustrate the operation of insertion sort on the array $\langle 4, 6, 2', 8, 5, 2\rangle$. Illustrate the last insertion in detail.

1b Illustrate merge sort on the sequence $A = \langle 4, 3, 6, 2, 9, 8, 4'\rangle$. Illustrate the last merge in detail. Give the result of each merge operation on the appropriate subarray in turn. The elements should be separated by commas.

1c Illustrate merge sort on the array $\langle 3; 41; 52; 26; 38; 57; 9; 49\rangle$. Show the last merge in detail.

1d Illustrate quicksort on the array $A = \langle 4, 3, 6, 2, 9, 8, 4\rangle$. We suppose that function partition() always selects the first item of the current subarray as

the pivot. Give the subarrays computed by each partition(A, p, r) call. The pivot must be distinguished by a '+' prefix. The items are separated by commas. For example 4,2,+5,8

1e Illustrate the operation of
function **partition** of quicksort on array $\langle 3; 41; 52; 26; 38; 57; 9; 49 \rangle$.
Let us suppose that the algorithm selects 41 as the pivot.

2. Let us consider the class Queue with its constructor, destructor, add(x:$\mathcal{T}$), rem():$\mathcal{T}$, and length():$\mathbb{N}$ operations.

$T(n) \in \Theta(n)$ for the destructor, and $T(n) \in \Theta(1)$ for all the other operations must be satisfied where n is the length of the queue.

Draw the UML box of the Queue, and the necessary structure diagrams to implement the operations in the following cases.

2a Let us represent the queue with a private S1L. (If the list is nonempty, an extra pointer refers to its last element.)

2b Let us represent the queue using a private one-way list with a trailer node. (One pointer refers to the first node, and an extra pointer refers to the trailer.)

2c Let us represent the queue using a private, cyclic one-way list with a header/trailer node. (The pointer identifying the list refers to its header/trailer node.)

2d Let us represent the queue using a private, cyclic one-way list without a header/trailer node. (If the list is nonempty, the pointer identifying it refers to its last node. If it is empty, it is identified by a $\oslash$ pointer.)

3a Pointer H refers to the header of an unsorted H1L. Write function remove_max(H) which removes an element with a maximal key from the list and returns the address of the element removed. $T(n) \in \Theta(n)$ where n is the length of the list.

3b Pointer H refers to the header of a non-increasingly sorted H1L. Write function sorted_insert(H, x) which inserts a new element with key x into the list, so that the list remains non-increasingly sorted. $MT(n) \in \Theta(n)$ where n is the length of the list.

3c Pointer L identifies an acyclic one-way list. Write function delete_list(L) which removes and deallocates (deletes) all the elements of the list. $T(n) \in \Theta(n)$ where n is the length of the list.

Name: ..... *Neptun code:* .....

4a Pointer H refers to the header of an unsorted C2L. Write function remove_max(H) which removes an element with a maximal key from the list and returns the address of the element removed. $T(n) \in \Theta(n)$ where n is the length of the list. List modifications must not be done directly, only through the procedures unlink(q), precede(q, r), and follow(p, q).

4b Pointer H refers to the header of a non-increasingly sorted C2L. Write function sorted_insert(H, x) which inserts a new element with key x into the list, so that the list remains non-increasingly sorted. $MT(n) \in \Theta(n)$ where n is the length of the list.

4c Pointer H refers to the header of a non-increasingly sorted C2L. Write function delete_list(H) which removes and deallocates (deletes) all the elements of the list. $T(n) \in \Theta(n)$ where n is the length of the list. List modifications must not be done directly, only through the procedures unlink(q), precede(q, r), and follow(p, q).

5a Let us suppose that L, H : E1* identify two strictly increasing H1Ls.

Write procedure union_intersection(L,H:E1*) rearranging the lists so that list L contains the sorted union of the keys found in the original lists, and list H contains the sorted intersection of the keys found in the original lists. Both lists must remain strictly increasing. Allocating and deallocating objects must be avoided in this program. And keys should not be copied from one object into another.

$MT_{\text{union\_intersection}}(n_L, n_H) \in O(n_L + n_H)$ is to be satisfied where n_L and n_H are the lengths of the lists.

To solve this problem, we can leave the common elements of the lists in list H, and move the other items of list H into list L. (No item will be moved from L to H.)

5b Let us suppose that L, H : E2* identify two strictly increasing C2Ls.

Write procedure union_intersection(L,H:E2*) rearranging the lists so that list L contains the sorted union of the keys found in the original lists, and list H contains the sorted intersection of the keys found in the original lists. Both lists must remain strictly increasing. Allocating and deallocating objects must be avoided in this program. And keys should not be copied from one object into another.

$MT_{\text{union\_intersection}}(n_L, n_H) \in O(n_L + n_H)$ is to be satisfied where n_L and n_H are the lengths of the lists.

Name: ..... *Neptun code:* .....

2 Set operations

Exercise 1 *Let us consider the class Queue with its constructor, destructor, add(x:$\mathbb{Z}$), rem():$\mathbb{Z}$, first():$\mathbb{Z}$, length():$\mathbb{N}$, and printQueue() operations.*

Let us represent the queue using a private, cyclic one-way list with a header/trailer node. (The pointer identifying the list refers to its header/trailer node.)

$T(n) \in \Theta(n)$ for the destructor and printQueue(), but $T(n) \in \Theta(1)$ for all the other operations must be satisfied where n is the length of the queue.

Define the class Queue with the operations and representation above. (Draw the UML box of the Queue, and the necessary structure diagrams to implement the operations.)

Let you have two queues, $Q1$ and $Q2$ containing strictly increasing sequences. Let you also have queue $Q3$ initially empty. Put the sorted intersection of the content of the queues $Q1$ and $Q2$ into $Q3$ while emptying $Q1$ and $Q2$. $MT(m,n) \in \Theta(m+n)$ where m is the original length of $Q1$, and n is the original length of $Q2$.

Exercise 2 *Let us consider the class Queue with its constructor, destructor, add(x:$\mathbb{Z}$), rem():$\mathbb{Z}$, first():$\mathbb{Z}$, length():$\mathbb{N}$, and printQueue() operations.*

Let us represent the queue with a private S1L. (If the list is nonempty, an extra pointer refers to its last element.)

$T(n) \in \Theta(n)$ for the destructor and printQueue(), but $T(n) \in \Theta(1)$ for all the other operations must be satisfied where n is the length of the queue.

Define the class Queue with the operations and representation above. (Draw the UML box of the Queue, and the necessary structure diagrams to implement the operations.)

Let you have two queues, $Q1$ and $Q2$ containing strictly increasing sequences. Let you also have queue $Q3$ initially empty. Put the sorted union of the content of the queues $Q1$ and $Q2$ into $Q3$ while emptying $Q1$ and $Q2$. $MT(m,n) \in \Theta(m+n)$ where m is the original length of $Q1$, and n is the original length of $Q2$.

Exercise 3 *Let us consider the class Queue with its constructor, destructor, add(x:$\mathbb{Z}$), rem():$\mathbb{Z}$, first():$\mathbb{Z}$, length():$\mathbb{N}$, and printQueue() operations.*

Let us represent the queue using a private one-way list with a trailer node. (One pointer refers to the first node, and an extra pointer refers to the trailer.)

$T(n) \in \Theta(n)$ for the destructor and printQueue(), but $T(n) \in \Theta(1)$ for all the other operations must be satisfied where n is the length of the queue.

Name: ..... *Neptun code:* .....

Define the class Queue with the operations and representation above. (Draw the UML box of the Queue, and the necessary structure diagrams to implement the operations.)

Let you have two queues, $Q1$ and $Q2$ containing strictly increasing sequences. Let you also have queue $Q3$ initially empty. Put the sorted difference of the content of the queues $Q1$ and $Q2$ into $Q3$ while emptying $Q1$ and $Q2$. $MT(m,n) \in \Theta(m+n)$ where m is the original length of $Q1$, and n is the original length of $Q2$.

Exercise 4 *Let us consider the class Queue with its constructor, destructor, add(x:$\mathbb{Z}$), rem():$\mathbb{Z}$, first():$\mathbb{Z}$, length():$\mathbb{N}$, and printQueue() operations.*

Let us represent the queue using a private, cyclic one-way list without a header/trailer node. (If the list is nonempty, the pointer identifying it refers to its last node. If it is empty, it is identified by a $\oslash$ pointer.)

$T(n) \in \Theta(n)$ for the destructor and printQueue(), but $T(n) \in \Theta(1)$ for all the other operations must be satisfied where n is the length of the queue.

Define the class Queue with the operations and representation above. (Draw the UML box of the Queue, and the necessary structure diagrams to implement the operations.)

Let you have two queues, $Q1$ and $Q2$ containing strictly increasing sequences. Let you also have queue $Q3$ initially empty. Put the sorted symmetric difference of the content of the queues $Q1$ and $Q2$ into $Q3$ while emptying $Q1$ and $Q2$. $MT(m,n) \in \Theta(m+n)$ where m is the original length of $Q1$, and n is the original length of $Q2$.

Exercise 5 *$H1$ and $H2$ are strictly increasing C2Ls. Write procedure union($H1, H2 : E2*$) which calculates the strictly increasing union of the two lists in $H1$ and empties $H2$.*

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n,m) \in \Theta(n+m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 6 *$H1$ and $H2$ are strictly increasing C2Ls. Write procedure intersection($H1, H2 : E2*$) which calculates the strictly increasing intersection of the two lists in $H1$ but does not change list $H2$.*

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *those list elements of $H1$ which turn out superfluous. We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n,m) \in \Theta(n+m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Name: ..... *Neptun code:* .....

Exercise 7 *$H1$ and $H2$ are strictly increasing C2Ls. Write procedure difference($H1, H2 : E2*$) which calculates the strictly increasing difference of the two lists in $H1$ but does not change list $H2$.*

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *those list elements of $H1$ which turn out superfluous. We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n, m) \in \Theta(n + m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 8 *$H1$ and $H2$ are strictly increasing C2Ls. Write procedure symmetricDifference($H1, H2 : E2*$) which calculates the strictly increasing symmetric difference of the two lists in $H1$ and empties list $H2$.*

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *those list elements of the lists which turn out superfluous. We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n, m) \in \Theta(n + m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 9 *$H1$ and $H2$ are strictly increasing H1Ls. Write procedure unionIntersection($H1, H2 : E1*$) which calculates the strictly increasing union of the two lists in $H1$ and the strictly increasing intersection of them in $H2$.*

We use neither memory allocation **(new)** *nor deallocation* **(delete)** *statements, nor assignment statements to data members. $MT(n, m) \in \Theta(n + m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 10 *$H1$ and $H2$ are strictly increasing H1Ls. Write procedure union($H1, H2 : E1*$) which calculates the strictly increasing union of the two lists in $H1$ and empties $H2$.*

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. $MT(n, m) \in \Theta(n + m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 11 *$H1$ and $H2$ are strictly increasing H1Ls. Write procedure intersection($H1, H2 : E1*$) which calculates the strictly increasing intersection of the two lists in $H1$ but does not modify list $H2$.*

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. $MT(n, m) \in \Theta(n + m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Name: ..... *Neptun code:* .....

Exercise 12 *$H1$ and $H2$ are strictly increasing H1Ls. Write procedure difference($H1, H2 : E1*$) which calculates the strictly increasing difference of the two lists in $H1$ but does not modify list $H2$.*

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. $MT(n,m) \in \Theta(n+m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 13 *$H1$ and $H2$ are strictly increasing H1Ls. Write procedure symmetricDifference($H1, H2 : E1*$) which calculates the strictly increasing symmetric difference of the two lists in $H1$ and empties list $H2$.*

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. $MT(n,m) \in \Theta(n+m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 14 *$H1$ and $H2$ are unsorted C2Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure union($H1, H2 : E2*$) which calculates the unsorted union of the two lists in $H1$ and empties $H2$.

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n,m) \in \Theta(n*m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 15 *$H1$ and $H2$ are unsorted C2Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure intersection($H1, H2 : E2*$) which calculates the unsorted intersection of the two lists in $H1$ but does not change list $H2$.

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *those list elements of $H1$ which turn out superfluous. We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n,m) \in \Theta(n*m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 16 *$H1$ and $H2$ are unsorted C2Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure difference($H1, H2 : E2*$) which calculates the unsorted difference of the two lists in $H1$ but does not change list $H2$.

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *those list elements of $H1$ which turn*

out superfluous. We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n, m) \in \Theta(n * m)$ where $n = lenght(H1)$ and $m = length(H2)$.

Exercise 17 *$H1$ and $H2$ are unsorted C2Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure symmetricDifference($H1, H2 : E2*$) which calculates the unsorted symmetric difference of the two lists in $H1$ and empties list $H2$.

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *those list elements of the lists which turn out superfluous. We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n, m) \in \Theta(n * m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 18 *$H1$ and $H2$ are unsorted C2Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure unionIntersection($H1, H2 : E2*$) which calculates the unsorted union of the two lists in $H1$ and the unsorted intersection of them in $H2$.

We use neither memory allocation **(new)** *nor deallocation* **(delete)** *statements nor assignment statements to data members. We rearrange the lists only with unlink(q), precede(q, r), follow(p, q). $MT(n, m) \in \Theta(n * m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 19 *$H1$ and $H2$ are unsorted H1Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure unionIntersection($H1, H2 : E1*$) which calculates the unsorted union of the two lists in $H1$ and the unsorted intersection of them in $H2$.

We use neither memory allocation **(new)** *nor deallocation* **(delete)** *statements, nor assignment statements to data members. $MT(n, m) \in \Theta(n * m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 20 *$H1$ and $H2$ are unsorted H1Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure union($H1, H2 : E1*$) which calculates the unsorted union of the two lists in $H1$ and empties $H2$.

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. $MT(n, m) \in \Theta(n * m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Name: ..... *Neptun code:* .....

Exercise 21 *$H1$ and $H2$ are unsorted H1Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure intersection($H1, H2 : E1*$) which calculates the unsorted intersection of the two lists in $H1$ but does not modify list $H2$.

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. $MT(n, m) \in \Theta(n * m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 22 *$H1$ and $H2$ are unsorted H1Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure difference($H1, H2 : E1*$) which calculates the unsorted difference of the two lists in $H1$ but does not modify list $H2$.

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. $MT(n, m) \in \Theta(n * m)$ where $n = lenght(H1)$ and $m = length(H2)$.*

Exercise 23 *$H1$ and $H2$ are unsorted H1Ls, with no duplicated key in $H1$, no duplicated key in $H2$. ($H1$ and $H2$ may have common keys.)*

Write procedure symmetricDifference($H1, H2 : E1*$) which calculates the unsorted symmetric difference of the two lists in $H1$ and empties list $H2$.

We use neither memory allocation **(new)** *nor assignment statements to data members. But deallocate* **(delete)** *the list elements which turn out superfluous. $MT(n, m) \in \Theta(n * m)$ where $n = lenght(H1)$ and $m = length(H2)$.*