CS231 Algorithms
Prof. Lyn Turbak
Wellesley College

Handout # 11
October 5, 2001

Problem Set 3
Due: Monday, October 15

Reading: Handout 10 (Induction, Loop Invariants, and List Sorting).

Required Problems: You should write up and turn in the solutions to the problems listed below: The points awarded per problem are given in brackets.

Problem 1 [25]: 2D-Searching

Call a 2-dimensional array of integers **2d-sorted** if each row is sorted from low to high value and each column is also sorted from low to high value. Here is an example of a 2d-sorted 8×8 array named B:

2	4	7	11	12	13	19	23
3	8	14	16	17	24	25	27
6	10	19	27	31	35	37	42
9	13	26	30	32	40	43	48
11	15	29	34	41	42	45	50
14	17	33	37	44	49	51	61
18	22	35	38	46	53	57	63
21	25	39	42	47	54	60	64

The notation $B[i, j]$ refers to the element in the ith row and jth column of B, where row and column indices start at 1. For example, $B[3, 2]$ is 10.

In this problem, we will consider various algorithms for determining whether a 2d-sorted $n \times n$ array contains a given number k. The algorithms will be expressed as a function `2D-Search` that takes a 2d-sorted $n \times n$ array and a number k as arguments and returns a boolean that is true if k is in the array and is false otherwise.

a. [4] Below is a straightforward iterative algorithm for searching a 2d-sorted $n \times n$ array for k.

```
2D-Search-a (A, k)
 for i ← 1 to n        ▷ number of rows
   for j ← 1 to n      ▷ number of columns
     if A[i, j] = k then return true   ▷ return exits the function immediately
 return false
```

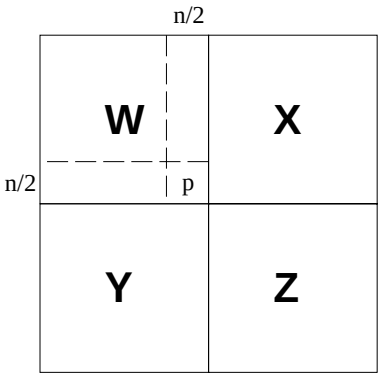
Give the worst-case running time of this algorithm as a function of n. Justify your answer.

b. [4] Suppose there is a function `Binary-Search(A, i, k)` that perforrms binary search on the ith row of `A` to determine if `k` is in the row; it returns true if `k` is in the row and false otherwise. Below is an algorithm that uses `Binary-Search` to search for `k` in a 2d-sorted $n \times n$ array.

```
2D-Search-b (A, k)
 for i ← 1 to n      ▷ number of rows
   if Binary-Search(A, i, k) then return true
 return false
```

Give the worst-case running time of this algorithm as a function of n. Justify your answer.

c. **[5]** Another way to search for k in a 2d-sorted $n \times n$ array is to use divide and conquer to generalize `Binary-Search` to work on a 2d-sorted array A. Here's the idea: suppose that p is the value in the array at $A[n/2, n/2]$. (See the following picture. For simplicity, assume that n is a power of 2 throughout this part.)



Then if $k = p$, the search terminates with true; if $k < p$, the search continues in the three square quadrants W, X, and Y; and if $k > p$, the search continues in the three square quadrants X, Y, and Z.

Give the worst-case running time of this algorithm as a function of n. Use the recursion tree method of solving recurrence equations to justify your answer.

d. **[10]** Develop an a $\Theta(n)$ worst-case running time algorithm that implements `2D-Search` on a 2d-sorted $n \times n$ array. Give pseudocode for your algorithm, and justify why it is $\Theta(n)$.

Hint: Any number k can be used to divide a 2d-sorted array into two parts: one consisting of the elements $\leq k$, and the other consisting of the elements $> k$. For example, the following figure shows a thick line separating the two parts for the sample array B and $k = 20$.

2	4	7	11	12	13	19	23
3	8	14	16	17	24	25	27
6	10	19	27	31	35	37	42
9	13	26	30	32	40	43	48
11	15	29	34	41	42	45	50
14	17	33	37	44	49	51	61
18	22	35	38	46	53	57	63
21	25	39	42	47	54	60	64

e. **[2]** Rank the algorithms from parts a through d by running time. An algorithm with running time f should appear above an algorithm with running time g if $f = \omega(g)$, and should appear on the same line if $f = \Theta(g)$.

Problem 2 [20]: Asymptotics of Assorted Sorts

For each of the four list sorting algorithms presented in Handout #10 (i.e., insertion sort, merge sort, quick sort, selection sort) give:

1. The asymptotic *worst-case* running time of the algorithm for a list with n elements.
2. The asymptotic *best-case* running time of the algorithm for a list with n elements.

Use Θ notation to express asymptotic running times. You should justify all of your running times, using recurrence equations where applicable.

Problem 3 [30]: A Splitting Headache

It is sometimes desirable to "split" the elements of a given list into two lists of nearly equal length. Here is a specification of a splitting function `split`.

Splitting Function Specification:

> Let `(cs,ds)` be the result of `split(es)`, where `cs`, `ds`, and `es` are all lists of elements. Then:
>
> **(split1)** $elts(cs) \cup_{\text{bag}} elts(ds) =_{\text{bag}} elts(es)$
> **(split2)** $0 \leq (length(\texttt{cs}) - length(\texttt{ds})) \leq 1$.
>
> Here, the *length* function returns the length of a list (i.e., the number of elements in it). E.g., $length([c_1, \ldots, c_k]) = k$.

a. **[8]** Here is a Haskell program for splitting a list:

```
split1 [] = ([], [])
split1 (x : xs) = (x : zs, ys)
  where (ys, zs) = split1 xs
```

Write a formal proof that `split1` satisfies the specification of `split` given above. This proof should be by induction. Carefully justify any use of the induction hypothesis.

b. **[7]** Prove the following fact about the Haskell `split1` function given in the previous part:

> If *sorted*(`hs`) and `(fs,gs)` is the result of `split(hs)`, then *sorted*(`fs`) and *sorted*(`gs`).

Here, *sorted* is the function defined in Handout #10 that determines if a list is sorted.

This proof should be by induction. Carefully justify any use of the induction hypothesis.

c. **[15]** The `split1` function given above is a *non-tail-recursive* function because there is pending computation to be performed after the recursive call to `split`. The following `split2` function uses an auxiliary *tail-recursive* `splitTail` function to implement an iterative approach to splitting a list:

```
split2 ws = splitTail ws [] []

splitTail [] ys zs = (ys,zs)
splitTail (x : xs') ys zs = splitTail xs' (x : zs) ys
```

Write a formal proof that `split2` satisfies the specification of a splitting function. Your proof should be based on the method of loop invariants. Recall that this requires the following steps:

1. Carefully state your loop invariants. These should express relationships between the parameters $\texttt{xs}_i$ ($= \texttt{x}_i$ `:` $\texttt{xs'}_i$ in the general case), $\texttt{ys}_i$, and $\texttt{zs}_i$ of the ith invocation of `splitTail` (i.e., `splitTail` $\texttt{xs}_i$ $\texttt{ys}_i$ $\texttt{zs}_i$).
2. Prove that the loop invariants hold the first time the loop is entered (i.e., `splitTail ws [] []`).
3. Prove that if the loop invariants are true at the beginning of an iteration (i.e., when a tail-recursive function is entered) they are true at the beginning of the next iteration (i.e., when the function is invoked tail-recursively in the body).
4. Prove that that the loop terminates (i.e., the base case of the tail recursive function is eventually reached).
5. Prove that the desired properties hold when the terminating state is reached.

Problem 4 [25]: Merge Sort Proofs

Recall the definition of the Haskell merge sorting function `msort` presented in class:

```
msort [] = []
msort [x] = [x]
msort xs = merge (msort ys, msort zs)
  where (ys,zs) = split xs

merge (ps, []) = ps
merge ([], qs) = qs
merge (p : ps, q : qs)
  | p <= q = p : (merge (ps, q : qs))
  | otherwise = q : (merge (p : ps, qs))
```

The definition of the `split` function is not given. You may assume that it is any correct implementation of the `split` specification given in the previous problem.

a. [20]

In this part, you are to write a complete proof that `msort` is a correct sorting algorithm. That is, you must show that `msort` satisfies the following `sort` specification:

> Let qs be the result of `sort(`ps`)`. Then:
>
> **(sort1)** $sorted(qs)$. I.e., if $qs = [q_1, q_2, \ldots, q_k]$, then $x_i \leq x_{i+1}$ for all $i \in [1 \,..\, k-1]$.
>
> **(sort2)** $elts(qs) =_{\text{bag}} elts(ps)$

To complete this proof, you must do the following:

1. Write a mathematical specification for the merging function that is sufficient for proving (below in part (3)) that `msort` is a correct sorting algorithm. What the merging function does in general is rather complex; your specification need only focus on the case where the two list arguments of the merging function are sorted.

2. Prove that `merge` satisfies your specification for the merging function from part (1). This proof should be by induction. Carefully justify any use of the induction hypothesis. You should consider the "size" of a pairs of lists `(ps, qs)` to be the sum of the lengths of `ps` and `qs`.
3. Prove that `msort` satisfies the `sort` specification. This proof should be by induction. Carefully justify any use of the induction hypothesis. You may assume that `merge` satisfies your merging function specification from part (1)

b. [5]

The definition of `msort` contains the line:

```
msort [x] = [x]
```

Would the definition of `msort` still be a correct sorting algorithm if this line were deleted from the definition? If yes, explain what would need to be changed in the above proof. If no, explain what goes wrong with the above proof.

Extra Credit 1 [10]: A Horse of a Different Color
Consider the following "proof" that all horses have the same color:

> Let S be a set of horses, and let $P[S]$ be true if all horses in S have the same color.
>
> *Base Cases* If S contains zero or one elements, then $P[S]$ is trivially true.
>
> *Inductive Case* Suppose S has n elements, and $P[X]$ is true for all sets $X \subseteq S$ containing i elements, where $0 \leq i \leq n-1$. Pick a horse h from S, and pick any two distinct subsets A and B of S that each (1) have $n-1$ elements and (2) contain h. By the inductive hypothesis, $P[A]$ and $P[B]$ are true; and since both A and B contain h, the color of the horses in $A \cup B = S$ must all be the same. So $P[S]$ is true.

Find and describe the bug in the above "proof".

Extra Credit 2 [20]: Dotopia

In a deep valley of one of the world's high mountain chains lies Dotopia, a village that has never had contact with the outside world. Dotopians have many interesting customs. For instance, all adult Dotopians celebrate the end of the day by gathering in village center at midnight. Another custom is that the passage to adulthood is marked by tattooing a permanent dot on the forehead, the color of which is randomly chosen to be red or blue.

But the strangest custom of all is this: if an adult Dotopian should ever learn with certainty the color of the dot on her forehead, that person is required to commit suicide in the village center at the daily midnight gathering on the day when she determines this fact. Understandably, Dotopians avoid looking at any reflective surfaces, and they are forbidden from ever talking about forehead dots with fellow villagers. It should be noted that all Dotopians are excellent logicians, and they always quickly and correctly deduce all possible conclusions from known facts.

One day, a lost mountain climber stumbles into Dotopia. He is befriended by the Dotopians, and learns all about their language and culture, including their strange custom. He is careful never to mention the color of people's dots, except for one fateful slip. At the midnight gathering on the day before he leaves Dotopia, he gives a speech to all the adult Dotopians in which he notes that he sees "both red dots and blue dots" in the crowd. He figures that this is safe to say, since he is not telling the Dotopians something that they cannot already see for themselves; there is more than one red-dotted person and more than one blue-dotted person in the crowd.

Nevertheless, some number of days after the speech, a portion of the Dotopian population commits suicide at the midnight gathering, and the rest of the population commits suicide at the very next midnight gathering. Explain why this tragedy happens, how it is a consequence of the speech, and why it never happened before the speech. In particular, what information in the speech precipitates the tragedy?
*H*int: The explanation is related to the inductive proof that Lake Day can never be a surprise.

Problem Set Header Page
Please make this the first page of your hardcopy submission.

CS231 Problem Set 3
Due Monday, October 15

Name:

Date & Time Submitted:

Collaborators (*anyone you worked with on the problem set*):

In the **Time** *column, please estimate the time you spend on the parts of this problem set. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the* **Score** *column when grading your problem set.*

Part	**Time**	**Score**
General Reading		
Problem 1 [25]		
Problem 2 [20]		
Problem 3 [30]		
Problem 4 [25]		
Extra Credit 1 [10]		
Extra Credit 2 [20]		
Total		