Fall 2006, CSE 565, Homework 1, solutions

Problem 2-3, correctness of Horner’s rule

a. Asymptotic running time is $O(n)$; it is actually a for loop ($a_i = a[i]$):

  
  H1  \text{    } y = 0;
  H2  \text{    } for \ (i = n; \ i \geq 0; \ i--)
  H3  \text{    } y *= x,
  H4  \text{    } y += a[i];

b. More naive pseudocode.

  N1  \text{    } y = 0;
  N2  \text{    } for \ (i = n; \ i \geq 0; \ i--)
  N3  \text{    } z = a[i];
  N4  \text{    } for \ (j = 0; \ j < n; \ i--) \text{    }
  N5  \text{    } z *= x;
  N6  \text{    } y += z;

The running time is $\Theta(\sum_{i=1}^{n} i) = \Theta(n^2)$. This is much slower than Horner’s rule.

The running time is $\Theta(\sum_{i=1}^{n} i) = \Theta(n^2)$. This is much slower than Horner’s rule.

c. Iterations of Horner’s rule use $i$ from $n$ down to 0; after iteration that uses $i = n$ we get $y = a_n$, this can be inductive basis for the claim that

  after iteration with $i = j$ we have

  \[ y = \sum_{k=j}^{n} a_k x^{k-j}. \]

Inductive step. We start iteration with $i = j$ with $y$ computed in iteration with $i = j + 1$, thus with

  \[ y = \sum_{k=j+1}^{n} a_k x^{k-j-1} \quad \text{executing} \quad \text{H3} \quad y = \sum_{k=j+1}^{n} a_k x^{k-j} \quad \text{executing} \quad \text{H4} \quad y = \sum_{k=j}^{n} a_k x^{k-j}. \]

d. When the algorithm terminates, we are after step with $i = 0$, and thus $y$ promised by the invariant is exactly the value we had to compute.
Problem 2-4, counting inversions.

**a.** (2,1), (3,1), (8,6), (8,1), (6,1).

**b.** In the worst case, every unordered pair of position is an inversion, and there are \( n(n-1)/2 \) such pairs. This happens when \( A[1] > A[2] > \cdots > A[n] \).

**c.** These numbers seem to be the same.

**d.** Divide and conquer schema: if the array is split into Left and Right, then we have three kinds of inversions:

- both \( A[i], A[j] \) inside Left,
- both \( A[i], A[j] \) inside Right,

We can find the first two kinds of inversion in recursive calls, and afterwards we can rearrange Left and Right and the existence of inversions of the third kind will be unaffected. Thus it is OK to sort Left and Right. Now consider inferences that we can make when we merge Left and Right, and there are \( \ell \) numbers remaining in the Left, and \( r \) in the right:

- if Left candidate is smaller, removing it does not remove any inversions
- if Right candidate is smaller, it is smaller than all elements of the Left, removing it removes \( \ell \) inversions.

We can use these inferences to count the inversions as we perform MERGE.

Conclusion: we can initialize the count of inversions to zero, run recursive MergeSort, and any time we perform a MERGE, the candidate from the Right is smaller and the Left has \( \ell \) elements remaining, we add \( \ell \) to the counter.
Problem 3-3, Ordering by asymptotic growth rates

Somehow nobody complained about $\lg^* n$ function.

Constant functions: $n^{1/\lg n} = 2^{\lg n/\lg n} = 2 \approx 1$

Next, functions with $\lg^* n$: $\lg(\lg^* n) << \lg^* n \approx \lg^*(\lg n) << 2^{\lg^* n}$.

The functions related to $\ln \ln n$ come next: $\ln \ln n$.

The functions related to $\ln n$ come next: $\sqrt{\ln n} << \ln n << \lg^* n$.

The functions related to $\lg n$ come next: $\sqrt{\lg n} << \ln n << \lg n$.

In between: $2^{\sqrt{\lg n}} = (2^{\lg n})^{1/\sqrt{\log n}} = n^{1/\sqrt{\log n}}$

Power functions:

$$\sqrt{2^{\lg n}} = (2^{\lg n})^{1/2} = n^{1/2} << 2^{\sqrt{\lg n}} << n = 2^{\lg n} << \lg(n!) \approx n \lg n <<$$

$$n^2 = 4^{\lg n} << n^3$$

Large functions:

$$n^{\lg \lg n} = 2^{\lg n \lg \lg n} = \lg n^{\lg n} << (3/2)^n << 2^n << n2^n << e^n << n! <<$$

$$(n + 1)! << 2^n << 2^{2^{n+1}}$$
Problem 4-2, Finding the missing integer

One can adapt PARTITION of QUICKSORT to partition sector A, p, r according to bit number k:

```
P1  i = p+1
P2  j = r
P3  while (1)
P4      do
P5          i++
P6      while (fetch(A[i],k))
P7          do
P8                  j--
P9      while (!fetch(A[i],k))
P10     if (i > j)
P11       return j
```

Example that we can generalize.

We start with sector A, 0, 21 that should contain all numbers from 0 to 21, but one is missing.

We partition according to the bit of the multiples of 16; we get 16 in return so now we have two sectors: A, 0, 16 and A, 16, 21.

16 numbers have zero on that bit position of 16 so none of these numbers is missing. The second sector has one missing number.

Now we have sector A, 16, 21 that should contain all numbers from 16 to 21 but one is missing. For all these numbers the bit of multiples of 8 is zero so we skip it. Now we partition according to the bit of multiples of 4, and we get two sectors: A, 16, 19 and A, 19, 21. The first sector should have 4 numbers and it has 3 so we continue the search there.

Now we have sector A, 16, 19 and we partition it according to the bit of 2’s, into A, 16, 18 and A, 18, 19; first sector should have two numbers, and it does, so we continue with the second.

Now we have sector A, 18, 19. The size of that sector is a power of 2, so when we partition it it may happen that there will be no number with one of the bit values, and in this case our PARTITION does not work. Ouch.

We can use an even simpler kind of partition in this case: move a number of 1 on the investigated position (now: multiples of 1, but it can be larger) to
the end of the sector if one is found; if one is indeed found we reduced the problem and if one is not found, we know what is missing. Thus if we start with $A, 16, 20$ we either learn that 20 is missing or we work with $A, 16, 19$.

I dimly recall that there exist a simpler method, but this one uses knowledge from the book (Hoare's partition).

Ouch, wrong problem!
Problem 4-2-2, solving a recurrence

First we simplify by having a new $T(n)$ that equals to the old one divided by $c$. The recurrence becomes

$$T(n) = T\left(\frac{1}{3}n\right) + T\left(\frac{2}{3}n\right) + n$$

For a continuous function this would be an equation

$$f(x) = f\left(\frac{1}{3}x\right) + f\left(\frac{2}{3}x\right) + x$$

Suppose that $f(x) = ax\ln x$ does satisfy that equation:

$$ax\ln x = a\frac{1}{3}x\ln\left(\frac{1}{3}x\right) a\frac{2}{3}x\ln\left(\frac{2}{3}x\right) + x$$

$$ax\ln x = a\frac{1}{3}x\ln\left(\frac{1}{3}x\right) a\frac{2}{3}x\ln\left(\frac{2}{3}x\right) + x$$

$$= a\frac{1}{3}x\left(\ln x + \ln \frac{1}{3}\right) + a\frac{2}{3}x\left(\ln x + \ln \frac{2}{3}\right) + x$$

$$= ax\ln x + a\left(\frac{1}{3}\ln \frac{1}{3} + \frac{2}{3}\ln \frac{2}{3} + \frac{1}{a}\right) = dx\ln x$$

The last equality is equivalent to

$$\frac{1}{3}\ln \frac{1}{3} + \frac{2}{3}\ln \frac{2}{3} + \frac{1}{a} = 0 \quad \equiv \quad a = \frac{3}{\ln 3 + 2\ln 1.5}$$

Because $n\ln n$ is bounded by a polynomial, we can use the identity that $T$ satisfies as a continuous function in the “proof” that $T(n) = an\ln n$; we know that neglecting the rounding effects and the basis conditions will not change $\Theta$ characteristics of $T$.

Fall 2006, CSE 565, Homework 2, solutions

Exercise 5.4-2

It is not easy to find an exact answer, so we will show a lower and upper bound of the form $c\sqrt{b}$, so the answer is $\Theta(\sqrt{b})$. 
Let $A_k$ be the event that we need at least $b$ tosses, and let $p_k$ be its probability.

The event $A_1$ is everything, and $p_1 = 1$: we must have at least one toss.

The event $A_{k+1}$ is the conjunction of $A_k$ and the failure of the $k$-th toss to land in a bin that already has such ball. Because among $b$ bins there were $k - 1$ full bins, $p_{k+1} = p_k \times (1 - \frac{k}{b})$.

The average number of tosses that we need is equal to

$$\sum_{k=1}^{\infty} p_k.$$

To provide a lower bound, consider $k < \sqrt{b}$:

$$(1 - \frac{1}{b})^{k+1} < 1 - \frac{k+1}{b} + \frac{(k+1)^2}{2b^2} \quad < \quad 1 - \frac{k}{b}$$

and thus $p_k + 1 > (1 - \frac{1}{b})^{k+1}$, and consequently,

$$p_k > (1 - \frac{1}{b})^{2+3+\cdots+k+1} < (1 - \frac{1}{b})^{(k+1)(k+2)/2}(1 - \frac{1}{b})^k \approx e^{-1}.$$

Therefore

$$\sum_{k=1}^{\infty} p_k > e^{-1} \sqrt{b}.$$

For the upper bound, observe that for $k > \sqrt{b}$ we have $p_{k+1} < (1 - \frac{\sqrt{b}}{b})p_k = (1 - \frac{1}{\sqrt{b}})p_k$, so $p_{\sqrt{b}+a} < (1 - \frac{1}{\sqrt{b}})^a$. Therefore we can estimate $\sum_{\sqrt{b}}^{\infty} p_k$ with $\sqrt{b}$ (probabilities cannot exceed 1), and $\sum_{\sqrt{b}}^{\infty} p_k$ with the geometric series $\sum_{0}^{\infty} (1 - \frac{1}{\sqrt{b}})^k = \sqrt{b}$. 


Exercise 5.4-6

We toss $n$ balls into $n$ bins. The probability that a particular bin remains empty is that every toss has missed it, so it equals $(1 - \frac{1}{n})^n \approx e^{-1}$.

The average number of empty bins equal to this particular bin is $e^{-1}$, so the overall average number empty bins is $ne^{-1}$.

The event that exactly one toss hit a particular bin has $n$ cases (one for every ball) that a ball hit that bin and all other balls missed, so it equals $n \times \frac{1}{n}(1 - \frac{1}{n})^{n-1} \approx e^{-1}$. 
Exercise 6.5-8

Algorithm to merge $k$ sorted lists into one. We have array $L[k]$ of pointers to the lists, and during the execution $L[i]$ points to the first element of the list number $i$ that was not moved to the merged list as yet.

This allows to maintain a priority queue of items 0 to $n - 1$ where the priority of item $i$ is $L[i]->key$. When $L[i]$ is at the root of the priority queue, we send $L[i]->key$ to the merged list and we replace $L[i]$ with $L[i]->next$. This can violate the heap property so we repair the heap.

Note that we change the priority only at the root, so we do not need cross-reference (no need to somehow find quickly the spot where we have to start the heap repair, we know where this spot is).
Problem 7-3

Stooge sort (not to be mistaken for Stougie sort, Stougie is a good Dutch computer scientist and an excellant billiard player).

\[
\text{StoogeSort}(A,i,j) \\
S1 \text{ if } (A[i] > A[j-1]) \\
S2 \quad \text{swap}(A,i,j-1) \\
S3 \text{ if } (j-i < 3) \\
S4 \quad \text{return} \\
S5 \quad k = (j-i)/3 \quad \text{// implicit rounding down} \\
S6 \quad \text{StoogeSort}(A,i,j-k) \\
S7 \quad \text{StoogeSort}(A,i+k,j) \\
S8 \quad \text{StoogeSort}(A,i,j-k)
\]

This code attempts to sort \(j - i\) numbers, and it makes sense only if \(j - i > 0\).

We claim that it indeed sorts. Basis: \(j - i < 3\), if \(j - i = 1\) we swap \(A[i]\) with itself, and if \(j - i = 2\) we accomplish the sort with conditional swap in line S2.

Digression: we do not have to swap when we make recursive calls, so a slightly more intelligent code would be

\[
\text{StoogeSort}(A,i,j) \\
S1 \text{ if } (j-i == 2) \\
S2 \quad \text{if } (A[i] > A[j-1]) \\
S3 \quad \text{swap}(A,i,j-1) \\
S4 \quad \text{return} \\
S5 \quad k = (j-i)/3 \quad \text{// implicit rounding down} \\
S6 \quad \text{StoogeSort}(A,i,j-k) \\
S7 \quad k = (j-i+1)/3 \quad \text{// implicit medium rounding} \\
S8 \quad \text{StoogeSort}(A,i+k,j) \\
S9 \quad k = (j-i+2)/3 \quad \text{// implicit rounding up} \\
S10 \quad \text{StoogeSort}(A,i,j-k)
\]

Correctness for 2 is again assured. In the inductive case let \(n = j - i\) and \(k_0\) be the value of \(k\) computed in line S9. The crux is that in line S8 we already have the top \(k_0\) numbers within the range of sorting, so these
numbers are placed correctly by this sort. Consequently, the last sort, in line S10, does not have to look at the top \( k_9 \) positions.

After the sort in line S6, the top \( k_9 \) numbers are either outside the range of this sort, and thus already in the range of S8 sort, or in the top \( k_9 \) positions of the range of S6.

To complete the proof, we need to see that the top \( k_9 \) positions of S6 fall in the sector of S8 sort.

If \( n = 3a \), then the range of S6 and the range of S8 have \( 2a \) numbers each, so they overlap on \( 2a + 2a - n = a \) numbers, and \( k_9 = a \).

If \( n = 3a + 1 \), then the range of S6 and the range of S8 have \( 2a + 1 \) numbers each, so they overlap on \( 2a + 1 + 2a + 1 - n = 2a + 1 \) numbers, and \( k_9 = a + 1 \).

If \( n = 3a + 2 \), then the range of S6 has \( 2a + 2 \) numbers and the range of S8 has \( 2a + 1 \) numbers, so they overlap on \( 2a + 2 + 2a + 1 - n = a + 1 \) numbers.

The recurrence for the number of comparisons is \( T(n) = 3T(\frac{2}{3}n) + 1 \), so by the Master theorem \( T(n) = \Theta(n^a) \) where \( a = \log_{3/2} 3 \). Because \( (3/2)^2 < 3 \) we have \( a > 2 \) and thus Stooge sort is slower than insert sort. As we call insert sort “stupid sort”, Howard, Fine and Howard deserve a fellowship in a Washington think tank, but they may have difficulty in an ordinary college.

Consider this a real-life like challenge. Someone wrote fantastically inefficient code and your task is to make it faster while modifying as little of it as possible. Replace one of the three calls with a faster function than sorting, prove the correctness, show that the new algorithm is faster than insert sort. However, suppose that because of some strange decisions in the past you have to sort \( A[n] \) recursively and can sort exclusively on fragments indicated in this schema. What you can do is replacing one of the recursive calls with some faster function. As this replacement takes place at every level of the recurrence,
Exercise 8.4-4, Bucket sorting by distance

We have points distributed in a unit circle and we have to sort them by
\[ d_i = \sqrt{x_i^2 + y_i^2}. \]
This is equivalent to sorting them by \( d_i^2 = x_i^2 + y_i^2 \) or according to \( d_i^c \) for any \( c > 0 \). The question is: which \( c \) will result in the uniform distribution, by area, of \((x_i, y_i)\) being converted to the uniform distribution of the values of \( d_i^c \).

The circle of radius \( \sqrt{1/2} \) has half of the area of the unit circle so points from that circle should contribute \( 1/2 \) of the value distribution. This means that sorting according to \( x_i^2 + y_i^2 \) serves our purpose exactly. This means that we split the points into those that have \( \lfloor n(x_i^2 + y_i^2) \rfloor = i, \ i = 0, \ldots, n - 1 \), sort each of these parts by insert sort and we combine the parts together.
Problem 9-2, Weighted Median

a. A normal median of a set of \( n \) nodes is defined as a number from the set that has fewer that \( n/2 \) smaller numbers and fewer than \( n/2 \) larger numbers in the set. When we fixed the weight of the set to 1 and of each element to \( 1/n \), this makes the weight of the smaller number smaller than \( 1/2 \) and the same with the weight of the larger numbers.

b. If our set is in a sorted array \( A[n] \), we can find the weighted median as follows:

\[
\text{F1 for (w = m = 0; w < 0.5; m++)}
\]
\[
\text{F2 w += A[i].weight;}
\]

c. We can adapt algorithms for finding order statistics to finding weighted median as follows: recursively, we deal with a set of total weight \( W \) and we want to find the maximum entry that has the total weight of smaller entries equal or lower than \( w \). We partition the set into \textit{Left}, \textit{pivot} and \textit{Right} as the original algorithm does. Then we calculate \( W_L \), the weight of the \textit{Left} and \( W_P \), the weight of the pivot. We have three cases: (i) \( W_L \geq w \), continue the search in \textit{Left} with parameters \( W_L \), \( w \); (ii) \( W_L < w \) and \( W_L+W_P \geq w \), return \textit{pivot}; (iii) \( W_L + W_P \geq w \), continue in \textit{Right} with parameters \( W - W_L - W_P \) and \( w - W_L - W_P \).

d. Given the coordinate \( x \) of the post office, our objective is to minimize \( \sum_{x_i \leq x} w_i(x - x_i) + \sum_{x_i > x} w_i(x_i - x) \). If we change the location by some \( \Delta \), the objective is changed by \( \Delta(\sum_{x_i \leq x} w_i - \sum_{x_i \geq x} w_i) \). It is easy to see that if \( x \) is not a weighted median than for some value of \( \Delta \) (it can be positive or negative) we obtain a better location \( x + \Delta \).

e. We find \( x \) that is a weighted median of \( x \)-coordinates and \( y \) that is a weighted median of \( y \)-coordinates. If the post office has one of its coordinates different that the respective weighted median, we can shift the location to a better one in the manner explained above.
Exercise 8.4-4, Bucket sorting by distance

We have points distributed in a unit circle and we have to sort them by $d_i = \sqrt{x_i^2 + y_i^2}$. This is equivalent to sorting them by $d_i^2 = x_i^2 + y_i^2$ or according to $d_i^c$ for any $c > 0$. The question is: which $c$ will result in the uniform distribution, by area, of $(x_i, y_i)$ being converted to the uniform distribution of the values of $d_i^c$.

The circle of radius $\sqrt{1/2}$ has half of the area of the unit circle so points from that circle should contribute $1/2$ of the distribution. This means that sorting according to $x_i^2 + y_i^2$ serves our purpose exactly. This means that we split the points into those that have $\lfloor n(x_i^2 + y_i^2) \rfloor = i$, $i = 0, \ldots, n-1$, sort each of these parts by insert sort and we combine the parts together.

Problem. Assume that the cost of QuickSort is $n \ln n$. (it is a “price”, not the number of comparisons etc.)

a. How much does the cost of QuickSort decrease, on the average, after executing a partition according to a uniformly random pivot?

Partition of cost $n$ reduce the problem instance of size $n$ elements to two parts of sizes that are uniformly distributed between 0 and $n-1$; in a particular instance we get $n \ln n$ reduced to $i \ln i + (n - i) \ln (n - i)$ so the (negative) change is

$$i \ln i + (n - i) \ln (n - i) - n \ln n =$$
$$i(\ln i - \ln n) + (n - i)(\ln (n - i) - \ln n) =$$
$$n \left[ \frac{i}{n} \ln \frac{i}{n} + \frac{n - i}{n} \ln \frac{n - i}{n} \right]$$

Let $x_i = i/n$, the average decrease is

$$2n \sum x_i \ln x_i (x_i - x_{i-1}) \approx 2n \int_0^1 x \ln x dx$$

We can use the formula $\int x^k \ln x dx = \frac{1}{k+1} x^{k+1} (\ln x - \frac{1}{k+1})$ which implies that $\int_0^1 x^k \ln x dx = \frac{1}{(k+1)^2}$. In this case, the average change is $2n \times \frac{1}{4} = \frac{n}{2}$.

To have a balance of the current cost ($n$ for the partition) with the decrease in the future cost we multiply the “cost function” by 2, thus it becomes $2n \ln n$.

b. How much does this cost decrease after partitioning according to the median?
The change is

\[ \frac{2n}{2} \ln \frac{n}{2} - n \ln n = n(\ln \frac{n}{2} - \ln n) = n \ln \frac{1}{2} \]

To have a balance of the current cost (\( n \) for the partition) with the decrease in the future cost we multiply the “cost function” by \( \frac{1}{\ln 2} = \log_2 e \) which has the effect of changing the base of the logarithm, so it becomes \( \log_2 n \).

c. How much does the average cost decrease after partitioning according to the median of a random sample that consist of 3 elements? Which of the variants seems to be the best one (knowing what you know about the cost of finding those pivots)?

The distribution of the median of 3 is not uniform; we pick the sample in \( C(n, 3) \approx \frac{n^3}{6} \) ways, and \( a_i \) is in the center of the sample in \( i(n-i) \) cases.

After introducing \( x = i/n \), we can express the probability of \( x_i \) corresponding to the median of 3 as

\[ \frac{i(n-i)}{n^3/6} = \frac{6i(n-i)}{n^3} = 6x_i(1-x_i)(x_i-x_{i-1}) = 6x(1-x)dx \]

This distribution is symmetric around 1/2, which means that \( x \) has the same probability as \( 1-x \) and we once again can express the average change as \( n \) times twice the integral of the change of the cost times the probability of the case:

\[ n2 \int_0^1 x \ln x \times 6x(1-x)dx = 12n \left[ \int_0^1 x^2 \ln xdx - \int_0^1 x^3 \ln xdx \right] = \\
12n \left( \frac{-1}{9} - \frac{-1}{16} \right) = \frac{7}{12}n \]

To have a balance of the current cost (\( n \) for the partition) with the decrease in the future cost we multiply the “cost function” by \( \frac{12}{7} \), thus it becomes \( \frac{12}{7}n \ln n \).