Problem 1. Assume that we have two algorithms for the bandersnatch problem:

- SlowSnatch uses $5 \cdot 2^n$ instructions to solve the bandersnatch problem of size n.
- FastSnatch uses $500n^3$ instructions to solve the bandersnatch problem of size n.

Assume that

- The hare, who thinks very quickly, does one trillion instructions per second.
- The tortoise, who thinks very slowly, does one million instructions per second.

(a) (i) How much time does the tortoise use to solve the bandersnatch problem on one thousand values using FastSnatch? Give your answer in seconds. You should calculate this by hand. Show your work.

Solution:

$$\frac{500(1000)^3}{10^6} = \frac{5(10)^{11}}{10^6} = 5(10)^5$$

(ii) How many days is this to two significant digits of accuracy? You may use a calculator.

Solution: 5.8

(b) (i) How much time does the hare use to solve the bandersnatch problem on one thousand values using SlowSnatch? Give your answer in seconds. You should calculate this by hand. Show your work. You can use the approximation $2^{10} \approx 1000$.

Solution:

$$\frac{5 \cdot 2^{1000}}{10^{12}} = \frac{5\left(2^{10}\right)^{100}}{10^{12}} \approx \frac{5\left(10^3\right)^{100}}{10^{12}} = \frac{5 \cdot 10^{300}}{10^{12}} = 5 \cdot 10^{288}$$

(ii) How many centuries is this to two significant digits of accuracy? You may use a calculator.

Solution:

$$1.6 \cdot 10^{279}$$

(c) How many times faster is the tortoise than the hare (in solving the bandersnatch problem)?

Solution: OMITTED.

Problem 2. Assume that we want to find the **length** of the **longest** Maximum Contiguous Sum in an array of length n. (The *length* of a contiguous sum is the number of values in the sum.) For each part you should just give the pseudo-code.

(a) Give a cubic ($\Theta(n^3)$ time, brute force) algorithm.

Solution:

```
M ← 0
L ← 0
for i = 1 to n do
    for j = i to n do
        S ← 0
        for k = i to j do
            S ← S + A[k]
        end for
        if (S>M) or ((S=M) and (j-i+1>L)) then
            M ← S
            L ← j-i+1
        end if
    end for
end for
```

(b) Give a quadratic ($\Theta(n^2)$ time) algorithm.

Solution:

```
M ← 0
L ← 0
for i = 1 to n do
    S ← 0
    for j = i to n do
        S ← S + A[j]
        if (S>M) or ((S=M) and (j-i+1>L)) then
            M ← S
            L ← j-i+1
        end if
    end for
end for
```

(c) Give a linear ($\Theta(n)$ time) algorithm.

Solution:

```
M ← 0
L ← 0
L_current ← 0
S ← 0
for i = 1 to n do
```

```
        S ← S+A[i]
        if S<0 then
            S ← 0
            L_current ← 0
        else
            L_current ← L_current + 1
        end if
        if (S>M) or ((S=M) and (L_current>L)) then
            M ← S
            L ← L_current
        end if
    end for
```

Assume that we have a computer in which computations can only be done in the registers: `R[0], R[1], R[2], ..., R[m-1]`. Memory accesses are "pipelined" so that memory locations `M[h], M[h+1], M[h+2], ... M[h+d-1]` can be moved to and from registers `R[g], R[g+1], R[g+2], ..., R[g+d-1]` at cost $\alpha f(h) + \beta d$, where α and β are constants and $f(h)$ is a (nondecreasing) function, all dependent on the actual computer. There is a computer dependent constant, s, such that the computer can read or write at most s values at a time (so d must be $\leq s$). To keep things simple, we assume that the number of registers is not much larger than s; perhaps $m = s + 16$. The two instructions are

`read(g,h,d)`, which copies `M[h], M[h+1], ... M[h+d-1]` to `R[g], R[g+1], ..., R[g+d-1]`
and

`write(g,h,d)`, which copies `R[g], R[g+1], ..., R[g+d-1]` to `M[h], M[h+1], ... M[h+d-1]`

Problem 1. Assume that an array of n numbers are in memory locations `M[1], M[2], M[3], ... M[n]`. Show how to implement bubble sort on the array. Give the pseudo-code. You can assume that a small number of index variables (such as i, j, and n) are in register locations $> s$, so that you can just use their names in your program and not worry about their register locations.

Solution:

```
for i = n downto 2 do
    if i≤s then read(1,1,i)
    for j = 1 to min(s,i)-1 do
        if R[j] > R[j+1]
            then R[j] ↔ R[j+1]
        end if
    end for
    write(1,1,i)
else
    R[0] ← R[min(s,i)]
    r ← min(s,i)
    while r+s < i do
        read(1,r,s)
        for j = 0 to s-1 do
            if R[j] > R[j+1]
                then R[j] ↔ R[j+1]
            end if
        end for
        write(r,0,s)
        j[0] ← R[s]
        r ← r+s
    end while
    read(r,1,i-r)
    for j = 0 to i-r do
        if R[j] > R[j+1]
```

```
                then R[j] ↔ R[j+1]
            end if
        end for
        write(r,0,i-r)
    end if
end for
```

Problem 2. Analyze the cost of memory accesses for your pseudo-code for the following four functions $f(h)$. Do not worry about comparisons, exchanges, etc. Just get the exact high order term involving α and the exact high order term involving β. Assume that $n >> s$. Your answers should be functions of α, β, s, and n.

Solution: We will ignore floors and ceilings in all of the calculations, which only affect lower order terms. All three parts have the same cost for the "β" part, so we will analyze it once and for all now:

$$\sum_{i=2}^{n}\sum_{k=1}^{i/s}\beta s \;=\; \beta\sum_{i=2}^{n}s\sum_{k=1}^{i/s}1 \;=\; \beta\sum_{i=2}^{n}si/s \;=\; \beta\sum_{i=2}^{n}i \;\sim\; \beta\frac{n^2}{2} \;=\; \frac{\beta n^2}{2}$$

(i) $f(h) = 1$.

Solution:

$$\sum_{i=2}^{n}\sum_{k=1}^{i/s}\alpha \;=\; \alpha\sum_{i=2}^{n}\sum_{k=1}^{i/s}1 \;=\; \alpha\sum_{i=2}^{n}i/s \;=\; (\alpha/s)\sum_{i=2}^{n}i \;\sim\; (\alpha/s)\frac{n^2}{2} \;=\; \frac{\alpha n^2}{2s}$$

(ii) $f(h) = h$.

Solution:

$$\sum_{i=2}^{n}\sum_{k=1}^{i/s}\alpha ks \;=\; \alpha\sum_{i=2}^{n}s\sum_{k=1}^{i/s}k \;\sim\; \alpha\sum_{i=2}^{n}s\frac{i^2}{2s^2} \;=\; \frac{\alpha}{2s}\sum_{i=2}^{n}i^2 \;\sim\; \frac{\alpha}{2s}\frac{n^3}{3} \;=\; \frac{\alpha n^3}{6s}$$

(iii) $f(h) = \ln(h)$.

Solution:

$$\sum_{i=2}^{n}\sum_{k=1}^{i/s}\alpha\ln(ks) = \alpha\sum_{i=2}^{n}\sum_{k=1}^{i/s}(\ln(k)+\ln(s)) \sim \alpha\sum_{i=2}^{n}\sum_{k=1}^{i/s}\ln k \sim \alpha\sum_{i=2}^{n}(i/s)\ln(i/s)$$
$$\sim \frac{\alpha}{s}\sum_{i=2}^{n} i\ln i \sim \frac{\alpha}{s}\frac{n^2\ln n}{2} = \frac{\alpha n^2\ln n}{2s}$$

Note that $\sum_{i=1}^{N}\ln i \sim N\ln N$ and $\sum_{i=1}^{N} i\ln i \sim \frac{N^2\ln N}{2}$, which we will prove later.

1. Consider the following insertion-sort-like algorithm: Sort the odd-indexed elements using insertion sort (leaving them in the odd-indexed locations). Sort the even-indexed elements using insertion sort (leaving them in the even-indexed locations). Sort all of the elements using (standard) insertion sort.

   (a) Write the pseudo code for this algorithm, *without* a sentinel.

Solution:

Insertion Sort (without sentinel)

```
for i = 3 to n step 2 do
    t ← A[i]
    j ← i-2
    while j>0 and A[j]>t do begin
        A[j+2] ← A[j]
        j ← j-2
    end while
    A[j+2] ← t
end for

for i = 4 to n step 2 do
    t ← A[i]
    j ← i-2
    while j>0 and A[j]>t do begin
        A[j+2] ← A[j]
        j ← j-2
    end while
    A[j+2] ← t
end for

for i = 2 to n do
    t ← A[i]
    j ← i-1
    while j>0 and A[j]>t do begin
        A[j+1] ← A[j]
        j ← j-1
    end while
    A[j+1] ← t
end for
```

   (b) Assume $n = 8$. What is the best-case number of comparisons? Just state the number and show your input. Otherwise, no justification needed.

Solution: Input: 1,2,3,4,5,6,7,8 Comparisons: 3+3+7=13

(c) Assume $n = 8$. What is the worst-case number of comparisons? Just state the number and show your input. Otherwise, no justification needed.

Solution: Input: 8,4,7,3,6,2,5,1 Comparisons: 6+6+16=28

(d) Calculate the number of comparisons the algorithm uses in the worst case for n even. Show your work.

Solution: The worst case is the list: $n, n/2, n-1, n/2-1, n-2, n/2-2, \ldots, n/2+3, 3, n/2+2, 2, n/2+1, 1$ The two sublists each take $\sum_{i=0}^{n/2-1} i = (n/2-1)(n/2)/2$ comparisons to sort.

For the final sort, the odd-indexed elements each take one comparison, except the first (which takes none). This is a total of $n/2-1$ comparisons.

The ith even-indexed element, except the first, takes $i+1$ comparisons. The first even element takes one comparison. Summing over all of the even-indexed elements gives

$$1+\sum_{i=2}^{n/2}(i+1) \;=\; 1+\sum_{i=1}^{n/2-1}(i+2) \;=\; 1+\sum_{i=1}^{n/2-1} i+\sum_{i=1}^{n/2-1} 2$$
$$=\; 1+(n/2-1)(n/2)/2+2(n/2-1) \;=\; 1+(n/2-1)(n/2+4)/2$$

Adding everything together gives

$$(n/2-1)(n/2)/2+(n/2-1)(n/2)/2+(n/2-1)+[1+(n/2-1)(n/2+4)/2]$$
$$=\; 1+(n/2-1)((3/2)n+6)/2 \;=\; \frac{3n^2+6n-16}{8}$$

1. Consider an array of size nine with the numbers in the following order $40, 20, 80, 60, 30, 90, 10, 70, 50$.

   (a) Form the heap using the algorithm described in class. Show the heap as a tree. Show the heap as an array. Exactly how many comparisons did heap creation use?

   Solution:

90	70	80	60	30	40	10	20	50

   $2+2+4+4 = 12$ comparisons

   (b) Start with the heap created in Part (a). Show the *array* after each element sifts down *after heap creation*. How many comparisons does each sift use? What is the total number of comparisons *after heap creation*?

   Solution:

80	70	50	60	30	40	10	20	90	4 comparisons
70	60	50	20	30	40	10	80	90	4 comparisons
60	30	50	20	10	40	70	80	90	4 comparisons
50	30	40	20	10	60	70	80	90	2 comparisons
40	30	10	20	50	60	70	80	90	2 comparisons
30	20	10	40	50	60	70	80	90	2 comparisons
20	10	30	40	50	60	70	80	90	1 comparison
10	20	30	40	50	60	70	80	90	0 comparisons

   Total of $4 + 4 + 4 + 2 + 2 + 2 + 1 + 0 = 19$ comparisons, after heap creation.

2. Assume that your machine has a built-in data structure that inserts an item into a priority queue in one step. There is no charge to remove an item. It has both a min-priority queue and a max-priority queue available. You can access the largest element for one comparison step in a max-priority queue, and similarly you can access the smallest element for one comparison step in a min-priority queue.

   This machine can sort a list in linear time: Insert all of the elements into a priority queue and then remove them (one at a time). This is n queue inserts and no comparisons, for list of size n.

   What if your priority queue is restricted to having size s? You can assume that your algorithm may have many different priority queues (but each can have size at most s). You can distinguish elements in the priority queue (perhaps by their original index in the list).

   (a) i. Design an efficient algorithm based on merge sort. You may describe it in high level English or in pseudo code, but the algorithm must be clear.

   Solution: Split the list into s sublists each of size n/s. Recursively sort each sublist. Then merge the s sublists using min-priority queue (of size s) to store one element of each sublist. It takes n inserts.

   ii. Analyze its time: Give the number queue inserts and number of comparisons as precisely as reasonably possible. At least give the high order term exactly. Your answer should be a function of the list size, n, and the queue size, s.

Solution: The recurrence is

$$M(n) = \begin{cases} sM(n/s) + n & n > s \\ n & \text{if } n \leq s \end{cases}$$

Using the tree method we find that for n a power of s, $M(n) = n \log_s n$.

(b) i. Design an efficient algorithm based on heapsort. You may describe it in high level English or in pseudo code, but the algorithm must be clear.

Solution: Use a heap where each node has s children. The children are stored in a max-priority queue of size s.

ii. Analyze its time: Give the number queue inserts and number of comparisons as precisely as reasonably possible. At least give the high order term exactly. Your answer should be a function of the list size, n, and the queue size, s.

Solution: Each sift takes (at most) $\log_s n$ inserts. We must do it for n elements giving a total $\sim n \log_s n$ inserts.

HW 5 Solutions

1a.

$$15n^{\lg 3} < 2n^2$$
$$\frac{15}{2} < n^{2-\lg 3}$$
$$n > (\frac{15}{2})^{\frac{1}{2-\lg 3}} = 128/35$$
$$\Rightarrow n = 129$$

1b.

$$129 * 64 = 8256$$

1c.

$$\lg x = 8256 \Rightarrow x = 2^{8256}$$
$$\log_1 0x = 8256 \log_1 02 = 2485.30$$
$$\Rightarrow 2486$$

2a.

$$T(3) = 18T(1) + 5(3)^2 + 4(3)$$
$$T(1) = 2$$
$$T(3) = 18(2) + 5(9) + 4(3) = 36 + 45 + 12 = 93$$

2bi. OMITTED (tree with branching factor of 18 where input size is divided by 3 at each subsequent level)

2bii.

$$\log_3 n$$

2biii.

$$18^{\log_3 n} = n^{log_3 18} \approx n^{2.63}$$

2biv.

$$2n^{2.63}$$

2bv.

$$\frac{n}{3^i}$$

2bvi.

$$5(\frac{n}{3^i})^2 + 4(\frac{n}{3^i})$$

2bvii.

$$18^i[5(\frac{n}{3^i})^2 + 4(\frac{n}{3^i})]$$

2bviii.

$$\sum_{i=0}^{\log_3 n-1} 18^i[5(\frac{n}{3^i})^2 + 4(\frac{n}{3^i})]$$

2bix.

$$= \sum_{i=0}^{\log_3 n-1} 18^i[\frac{5n^2}{9^i} + \frac{4n}{3^i}]$$

$$= 5n^2 \sum_{i=0}^{\log_3 n-1} 2^i + 4n \sum_{i=0}^{\log_3 n-1} 6^i$$

$$= 5n^2 \frac{1-2^{\log_3 n}}{1-2} + 4n\frac{1-6^{\log_3 n}}{1-6}$$

$$= 5n^2(n^{\log_3 2} - 1) + \frac{4n}{5}(n^{1+\log_3 2} - 1)$$

$$= \frac{29}{5}n^{2.63} - 5n^2 - \frac{4n}{5}$$

2bx.

$$= \frac{29}{5}n^{2.63} - 5n^2 - \frac{4n}{5} + 2n^{2.63}$$

$$= \frac{39}{5}n^{2.63} - 5n^2 - \frac{4n}{5}$$

3a.

```
def oddEvenSort(A, start, inc):
    if start + inc > n:
        return
    oddEvenSort(A, start, 2*inc)
    oddEvenSort(A, start + inc, 2*inc)

    for i = (start + inc) to n by inc:
        t = A[i]
        j = i - inc
        while j > 0 and A[j] > t do:
            A[j + inc] = A[j]
            j = j - inc
        A[j+inc] = t

oddEvenSort(A, 1, 1)
```

3b.
In the worst case after the odd even sort, insertion sort does: (from HW3):
Array position 1: no comparisons
Array position 2: 1 comparison
Array position 3: 1 comparison
Array position 4: 3 comparisons
Array position 5: 1 comparison
Array position 6: 4 comparisons
Array position 7: 1 comparison
Array position 8: 5 comparisons

From this pattern, ignoring the first 2, odd positions do 1 comparison and even positions do $\frac{i}{2} + 1$ comparisons
So we get:

$$\begin{aligned} & 0 + 1 + \sum_{i=1}^{n/2-1} 1 + \sum_{i=1}^{n/2-1} i + 2 \\ &= 1 + (\frac{n}{2} - 1) + 2((\frac{n}{2} - 1)) + \frac{(n/2-1)(n/2)}{2} \\ &= 1 + 3(\frac{n}{2} - 1) + (n/2 - 1)\frac{n}{4} \\ &= 1 + (\frac{n}{2} - 1)(\frac{n}{4} + 3) \\ &= \frac{n^2}{8} + \frac{5n}{4} - 2 \end{aligned}$$

So the complete recurrence is:

$$T(n) = 2T(n/2) + \frac{n^2}{8} + \frac{5n}{4} - 2$$
$$T(1) = 0$$

Problem 1. Consider an array of size eight with the numbers $50, 70, 10, 20, 60, 40, 80, 30$. Assume you execute quicksort using the version of partition from CLRS. Note that in this algorithm an element might exchange with itself (which counts as one exchange).

(a) Show the array after the first partition. How many comparisons and exchanges are used?

Solution:

$$10,\ 20,\ 30,\ 70,\ 60,\ 40,\ 80,\ 50\ .$$

Comparisons: 7

Exchanges: 3

(b) Show the left side after the next partition. How many comparisons are used? How many exchanges?

Solution:

$$10, 20$$

Comparisons: 1

Exchanges: 2

(c) Show the right side after the next partition on that side. How many comparisons are used? How many exchanges?

Solution:

$$40,\ 50,\ 70,\ 80,\ 60.$$

Comparisons: 4

Exchanges: 2

(d) What is the total number of comparisons in the entire algorithm? What is the total number of exchanges in the entire algorithm?

Solution: Comparisons: 15

Exchanges: 9

Problem 2. We are going to derive the average number of moves for quicksort using a somewhat unusual partitioning algorithm. We partition on the first element. Take it out. Look for the right most element that is smaller and place it in the first position (which is the newly opened position on the left side). Look for the left most element that is larger and place it in the newly opened position on the right side. Starting from there look for the right most element that is smaller and place it in the the newly opened position on the left side. Starting from there look

for the left most element that is larger and place it in the newly opened position on the right side. Continue in this fashion until the pointers cross. Finally, put the partition element into the hole, which is its final position in the sorted array.

(a) Assume that the partition element ends up in position q.

i. What is the probability that an element originally to the left (of position q) went to the right (of position q)?

Solution: There are $q-1$ positions to the left of position q and $n-q$ to the right. So the probability that an element originally on the left goes to the right is $(n-q)/[(q-1)+(n-q)] = (n-q)/(n-1)$ (not counting the partition element itself).

ii. What is the expected number of elements originally to the left that go to the right?

Solution: Counting the element originally in position q as being on the left, and not counting the pivot element, there are $q-1$ elements on the left. So the expected number of moves to the right is $(q-1)(n-q)/(n-1)$.

iii. What is the probability that an element originally to the right went to the left?

Solution: Similarly to the above reasoning the probability that an element originally on the right goes to the left is $(q-1)/(n-1)$.

iv. What is the expected number of elements originally to the right that go to the left?

Solution: There are $(n-q)$ elements on the right, so the expected number of moves to the left is $(q-1)(n-q)/(n-1)$.

v. What is the total expected number of moves (for partition)?

Solution: Finally, the pivot element is moved out, and then into position q. So there are two extra moves.
The total is

$$\frac{(q-1)(n-q)}{n-1} + \frac{(q-1)(n-q)}{n-1} + 2 \;=\; \frac{2(q-1)(n-q)}{n-1} + 2$$

Notice, that this formula holds even for $q = 1$.

(b) Write a recurrence for the expected number of moves (for quicksort).

Solution:

$$T(n) = \sum_{q=1}^{n} \frac{1}{n} \left[T(q-1) + T(n-q) + \frac{2(q-1)(n-q)}{n-1} + 2 \right]$$

$T(0) = T(1) = 0$.

(c) Simplify the recurrence, but do not solve it.

Solution:

$$T(n) = \frac{1}{n}\sum_{q=0}^{n-1}\left[2T(q) + \frac{2(q-1)(n-q)}{n-1} + 2\right]$$

$T(0) = T(1) = 0$.

(d) To keep the calculations simple, assume you got the following recurrence:

$$T(n) = \frac{2}{n}\left[\sum_{q=1}^{n-1} T(q) + q - \frac{q^2}{n}\right]$$

Use constructive induction to show that $T(n) \leq an\ln n$. Try to get a good value of a. [You should NOT actually get this exact recurrence for Part (c).]

Solution: We claim that $T(n) \leq an\ln n$ for some constant a and $n \geq 1$. Proof by constructive induction.

Base case: $n = 1$: $T(1) = 0$ and $a \cdot 1 \cdot \ln 1 = 0$.

Induction step: Assume it holds for all positive integers less than n. Then

$$\begin{aligned}
T(n) &= \frac{2}{n}\sum_{q=1}^{n-1}[T(q) + q - \frac{q^2}{n}] \\
&\leq \frac{2}{n}\sum_{q=1}^{n-1}[aq\ln q + q - \frac{q^2}{n}] \qquad \text{by the Induction Hypothesis} \\
&= \frac{2}{n}\sum_{q=1}^{n-1}[aq\ln q + q - \frac{q^2}{n}] \\
&\leq \frac{2}{n}\int_1^n [aq\ln q + q - \frac{q^2}{n}]dq \qquad \text{by the Integral Bound} \\
&= \frac{2}{n}[\frac{aq^2}{2}\ln q - \frac{aq^2}{4} + \frac{q^2}{2} - \frac{q^3}{3n}]\,|_1^n \\
&= \frac{2}{n}[(\frac{an^2}{2}\ln n - \frac{an^2}{4} + \frac{n^2}{2} - \frac{n^3}{3n}) - (\frac{a1^2}{2}\ln 1 - \frac{a1^2}{4} + \frac{1^2}{2} - \frac{1^3}{3n})] \\
&= an\ln n - \frac{an}{2} + n - \frac{2n}{3} - 0 + \frac{a}{2n} - \frac{1}{n} - \frac{2}{3n^2} \\
&= an\ln n + (\frac{1}{3} - \frac{a}{2})n + (\frac{a}{2} - 1)\frac{1}{n} - \frac{2}{3n^2} \\
&\leq an\ln n \qquad \text{for } a = \tfrac{2}{3}.
\end{aligned}$$

So the number of moves is

$$T(n) \leq \frac{2}{3}n\ln n \;=\; \frac{2}{3}(\ln 2)\lg n \;\approx\; .46n\lg n$$

(e) Floyd's version of heapsort makes about $n \lg n$ moves. How does quicksort compare?

Solution: Heapsort makes about $n \lg n$ moves, so quicksort makes slightly less than half as many on average.