Homework #1

(Due: Oct 11)

Task 1. [50 Points] Analysis of a Loop-based Algorithm

Figure 1 below shows a non-recursive loop-based algorithm.

```
ALGO( n )
 1. allocate array P[1 : n]
 2. for i ← 1 to n do
 3.     P[i] ← 0
 4. for i ← 2 to n do
 5.     j ← 2i
 6.     while j ≤ n do
 7.         P[j] ← P[j] + 1
 8.         j ← j + i
 9. for i ← 2 to n do
10.     if P[i] = 0 then print i
11. free P
12. return
```

Figure 1: A loop-based algorithm.

(*a*) **[10 Points]** What computational problem does the algorithm solve? Justify your answer.

(*b*) **[20 Points]** Give a tight bound on the worst-case running time of the algorithm. Show your analysis.

(*c*) **[20 Points]** Suppose that we replace the statement in line 5 with $j \leftarrow 2i^2$ and the one in line 8 with $j \leftarrow j + i^2$. Now give a tight bound on the worst-case running time of this modified version of the algorithm. Show your analysis.

Task 2. [90 Points] Improved Insertion Sort

Figure 2 shows IMPROVED-INSERTION-SORT – an improved version of the insertion sort algorithm that we saw in the class. We intentionally avoided explaining the algorithm and adding comments to the pseudocode in order to make life difficult for you. Your task will be to understand, explain, analyze, and improve it even further.

```
IMPROVED-INSERTION-SORT( A, n )
Input: An array A containing n numbers, where n is a positive integer.
Output: The numbers in A rearranged in nondecreasing order of value.
 1. m ← ⌈√n⌉
 2. allocate arrays block_data[1 : m][1 : 2m], block_size[1 : m], block_left[1 : m]
 3. k ← 1, rightmost_block ← k, next_free_block ← k + 1
 4. block_data[k][1] ← A[1], block_size[k] ← 1, block_left[k] ← 0
 5. for j ← 2 to n do
 6.     k ← rightmost_block
 7.     while k > 1 and block_data[k][1] > A[j] do
 8.         old_k ← k, k ← block_left[k]
 9.     if block_size[k] = 2m then
10.         q ← next_free_block, next_free_block ← q + 1
11.         for i ← 1 to m do
12.             block_data[q][i] ← block_data[k][m + i]
13.         block_size[k] ← m, block_size[q] ← m, block_left[q] ← k
14.         if k = rightmost_block then rightmost_block ← q
15.         else block_left[old_k] ← q
16.         if block_data[q][1] ≤ A[j] then k ← q
17.     l ← block_size[k], block_data[k][l + 1] ← A[j], block_size[k] ← l + 1
18.     while l > 0 and block_data[k][l] > block_data[k][l + 1] then
19.         block_data[k][l + 1] ↔ block_data[k][l], l ← l − 1
20. k ← rightmost_block, i ← n
21. while k > 0 do
22.     for l ← block_size[k] downto 1 do
23.         A[i] ← block_data[k][l], i ← i − 1
24.     k ← block_left[k]
25. free block_data, block_size, block_left
26. return
```

Figure 2: An improved insertion sort algorithm.

(*a*) [**20 Points**] Performance of which aspect/part of the original insertion sort algorithm does IMPROVED-INSERTION-SORT try to improve? How and why does this improved version work? Give a clear and detailed explanation.

(*b*) [**30 Points**] Show that the worst-case running time of IMPROVED-INSERTION-SORT on an input of size n is $\mathcal{O}\left(n^{1+\frac{1}{2}}\right)$ instead of the $\mathcal{O}\left(n^2\right)$ bound we proved in the class for the original insertion sort algorithm. Are there inputs on which the improved algorithm will take $\Theta\left(n^{1+\frac{1}{2}}\right)$ time in the worst case? Justify your answer.

```
MERGE-QUICK-SORT( A, p, r )
Input: A subarray A[p : r] containing r − p + 1 numbers, where p ≤ r.
Output: The numbers in A[p : r] rearranged in nondecreasing order of value.
1. if p < r then
2.     q ← ⌊(p+r)/2⌋
3.     QUICK-MERGE-SORT ( A, p, q )
4.     QUICK-MERGE-SORT ( A, q + 1, r )
5.     MERGE ( A, p, q, r )
6. return
```

```
QUICK-MERGE-SORT( A, p, r )
Input: A subarray A[p : r] containing r − p + 1 numbers, where p ≤ r.
Output: The numbers in A[p : r] rearranged in nondecreasing order of value.
1. if p < r then
2.     q ← PARTITION ( A, p, r )
3.     MERGE-QUICK-SORT ( A, p, q − 1 )
4.     MERGE-QUICK-SORT ( A, q + 1, r )
5. return
```

Figure 3: A recursive sorting algorithm that uses merging and partitioning at alternate levels of recursion.

(*c*) [**10 Points**] What is the best-case running time of IMPROVED-INSERTION-SORT on an input of size n? Are there inputs on which it achieves the best-case asymptotic running time? Justify your answer.

(*d*) [**30 Points**] Explain how you can improve the worst-case running time of the algorithm even further to $\mathcal{O}\left(n^{1+\frac{1}{3}}\right)$ by extending the idea used by IMPROVED-INSERTION-SORT.

Task 3. [**50 Points**] QUICK-MERGE-SORT **and** MERGE-QUICK-SORT

Figure 3 shows a recursive sorting algorithm that is a hybrid of two other sorting algorithms we analyzed in the class, namely, merge sort and quicksort. It uses merging and partitioning at alternate levels of recursion.

(*a*) [**30 Points**] We proved in the class that the worst-case running time of merge sort is $\mathcal{O}(n \log n)$, while that of quicksort is $\mathcal{O}\left(n^2\right)$, where n is the number of items to sort. Then what is the worst-case running time of QUICK-MERGE-SORT? Show your analysis.

(*b*) [**10 Points**] Is the bound you proved in part (*a*) tight? Justify your answer.

(*c*) [**10 Points**] What is the worst-case running time of MERGE-QUICK-SORT? Justify your answer.

Task 4. [60 Points] Sort by Growth Rate

Arrange the following functions in nondecreasing order of growth rate (show necessary derivations), and mark the ones that are asymptotically equal. Identify the function(s) that cannot be placed in any specific location in the ordering and explain why.

Assume that the 'log' function has base 2 unless explicitly mentioned otherwise and that c is a nonnegative constant.

$$n^{\log\left(\sqrt[\log n]{\log n}\right)},\quad (\log n)^{3\left\lceil \frac{n}{3}\right\rceil - n},\quad 5^{\log_3 n},\quad \sqrt{2}+\sin n+\cos n,\quad \frac{\log n}{c+\frac{1}{n}},\quad (n!)^{\frac{1}{n}},\quad c^n,\quad n^5,\quad \log_3 n,\quad 5^{\log n}$$

Note: To prove that $f_1(n), f_2(n), \ldots, f_k(n)$ is a correct ordering of the functions $\{f_1(n), f_2(n), \ldots, f_k(n)\}$, it suffices to show that $f_i(n) = \mathcal{O}\left(f_{i+1}(n)\right)$, for $1 \leq i \leq k-1$.