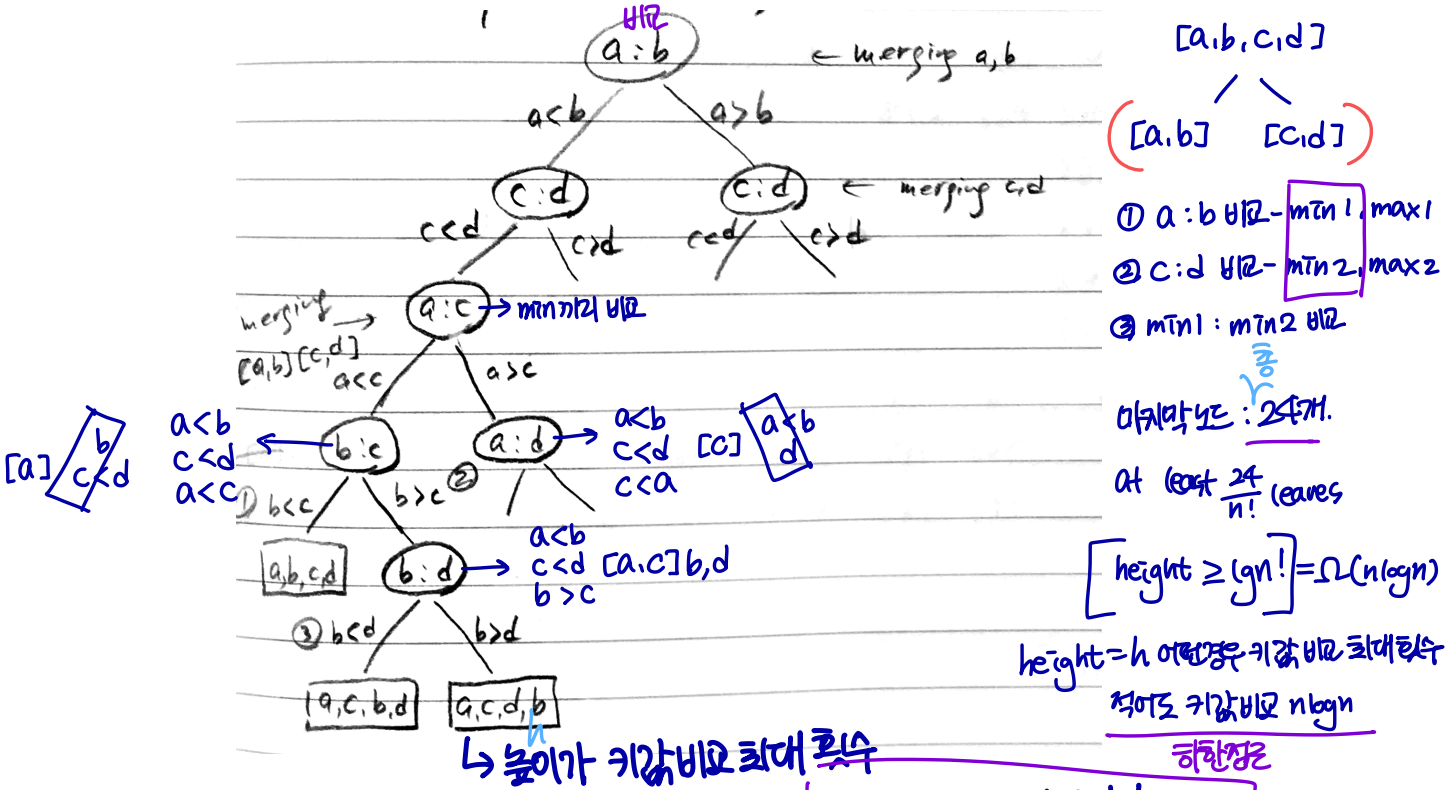
3 Complexity of Sorting

3.1 Lower Bound for Comparison-Based Sorting

We have shown that mergesort runs in $O(n \log n)$ steps. Now we will show that it takes at least $\Omega(n \log n)$ steps to sort. To do that, we consider an object called *comparison tree*.

Suppose that we have four arbitrary but distinct elements $[a, b, c, d]$ and mergesort them. We want to trace all the comparisons that occur during the sorting procedure in all the possible configurations about the four elements. Recall how mergesort works and we see that the first comparison occurs is between a and b, which is denoted by $a : b$. [[More explanation on the construction in class :) ..]] The following is the comparison tree of mergesort on $[a, b, c, d]$.



A comparison tree is constructed as follows:

(a) An inner node represents a comparison $x : y$.

(b) The result of $x : y$ is either $x < y$ or $x > y$. For each result, corresponds an edge. So, a comparison tree is a binary tree.

(c) If a comparison $z : w$ occurs after the comparison $x : y$ and the result $x < y$, then $z : w$ is the left child. If $z : w$ occurs as the result of $x > y$, then it is the right child.

(d) After the last comparison, depending on the result of it, the sorting results in a permutation of the input, which becomes a leaf of the comparison tree.

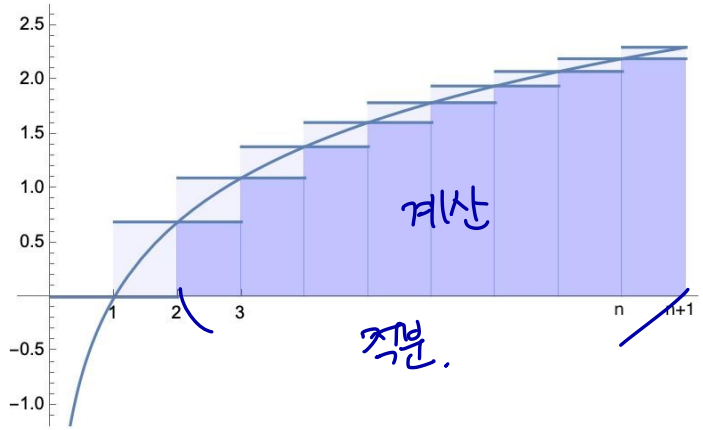
Try to construct the above comparison tree yourself by recalling how mergesort works!

For an input $[a_1, \ldots, a_n]$ of n elements, a *correct* sorting algorithm has a comparison tree with at least $n!$ leaves, since all possible inputs must be correctly sorted. A path from the

root to a leaf corresponds to the execution of the sorting procedure for an input, and the path length is the number of key comparisions occurred. Therefore, the height of the comparison tree corresponds to the maximum possible number of key comparisons that occur for the algorithm to sort an arbitrary n distinct elements.

Theorem 5. *A comparison-based sorting algorithm takes $\Omega(n \log n)$ comparisons.*

Proof. The height of a binary tree with N leaves is at least $\lceil \lg N \rceil$. Since there are at least $n!$ leaves on the comparison tree, its height must be at least $\lceil \lg n! \rceil$, which is $\Theta(n \lg n)$. Now, in the worst case, the number of necessary comparisons is the height. (In fact, the average depth is also $\Omega(\lg n!) = \Omega(n \lg n)$. Prove this.) Usually, we use Stirling's formula $n! \sim \sqrt{2\pi n}(n/e)^n$ to see that the height is at least of order $n \log n$. However, there is very elementary estimation. Since $\log n! = \sum_{1 \le k \le n} \log k$, we make an estimation of the summation with an integration of $\log x$. Consider the following:



The curve is for $\log x$, and the area covered by the lightly shaded bars ranging from 1 to n is $\sum_{k=1}^{n} \log k$. Also the blue shaded bars ranging from 2 to $n+1$ also gives the same summation. The fisrt area is larger than the integration $\int_1^n \log x dx$, and the second (shaded) area is smaller than the integration $\int_1^{n+1} \log x dx$. Therefore, we have

$$n \log n - n + 1 = \int_1^n \log x dx \le \sum_{k=1}^{n} \log k \le \int_1^{n+1} \log x dx = (n+1)\log(n+1) - n,$$

which shows that $\sum_1^n \log k = \Theta(n \log n)$. □

3.2 Radix Sort

[[First, about the name: bucket sort, radix sort, counting sort, and distribution sort.]] See [CLRS, chapter 8].

Suppose that we have the following twenty three-digit decimal numbers:

675 369 272 095 110 853 299 571 230 474 533 278 197 693 423 168 501 068 502 187

Set up ten buckets for each digit and put the numbers in the bucket according to the last digit.

(First phase) → 일의 자리수 정렬

0	1	2	3	4	5	6	7	8	9
230	571	272	853	474	675		197	278	369
	501	502	533		095		187	168	299
			693					068	
			423						

→ 순서대로 넣기.

Now as the second phase, put the numbers in the buckets according to the second digit, starting from the 0th bucket to the 9th and in the same bucket from the top to bottom (these orders are important).

(Second phase) → 십의 자리수 정렬

0	1	2	3	4	5	6	7	8	9
501		423	230		853	168	571	187	693
502			533			068	272		095
						369	474		197
							675		299
							278		

→ 순서대로 백의 자리수 정렬

The third phase repeat the same for the first digit.

(Third phase) → 백의 자리수 정렬

0	1	2	3	4	5	6	7	8	9
068	168	230	369	423	501	675		853	
095	187	272		474	502	693			
	197	278			533				
		299			571				

→ 정렬완료

Now take the numbers by the same manner from the 0th bucket, and see that the numbers are sorted correctly.

In general, suppose that we have N r-ary numbers of d-digits. Then, we can sort them in d phases. Each phase takes $O(N)$ steps. Therefore, radix sort runs in $O(d \cdot N)$ time. We need r buckets to run this algorithm. But with an appropriate data structure, for example, linked lists for each bucket, the memory requirement is $O(N)$.

자리수 × 개수만큼 하나씩 옮겨

Since d is usually fixed, radix sort runs in $O(N)$ time. However, we proved that sorting requires $\Omega(N \log N)$. What's wrong? Notice that radix sort does not use "comparison operation." Therefore, Theorem 5 does not apply in this case. But then, can we use radix sort for any sorting problem? No! Radix sort cannot be used for, for example, sorting problems where keys are floating point numbers. Now then, another important question: can comparison-based sorting, like quicksort and mergesort, be used for any sorting problem? Yes! If a set of data is sorted, we assume a linear order on the set, which means that any two elements are comparable. So, comparison-based sorting is applicable on any sorting problem, and, therefore, Theorem 5 is important.

비교 X → nlogn X

데이터 개수 N — r진법 r-ary — d단계 d-digit (ex) 10진수 : 바구니 0-9 10개. (* r-ary 바구니가 r개인거임)

각 단계 N번 데이터 읽음 → d단계니까 O(d·N)

↳ radix는 비교 X.

키값비교는 nlogn이 최소.

비교기반정렬 알고리즘 — nlogn이 최고

merge, quick — O(nlogn)임

radix — 비교 X O(n)