1. Maximum numbers of inversions

Base case: When n=2. Substitute n=2 into $I_{max:n}=I_{max:2}=\frac{n(n-1)}{2}=\frac{2(2-1)}{2}=1$. We have shown $I_{max:2}=1$, therefore it is true for n=2.

Inductive assumption: Assume that it is true for n=k. That is, $I_{max:k}=\frac{k(k-1)}{2}$.

Inductive proof: $I_{max:k+1}=\frac{(k+1)(k+1-1)}{2}=\frac{k(k+1)}{2}$. So it is true for n=k+1.

Therefore the statement is true for size n≥2 of a list A_n.

The average number of inversions in an array of n randomly selected elements is $\frac{n(n-1)}{4}$. This is because the minimum number of inversions (best case) is 0, and the maximum (worst case) is $\frac{n(n-1)}{2}$, hence the average would be the midpoint of these two values, $\frac{n(n-1)}{4}$.

Average numbers of inversions

Base case: When n=2. Substitute n=2 into $I_{ave:n}=I_{ave:2}=\frac{n(n-1)}{4}=\frac{2(2-1)}{4}=0.5$. We have shown $I_{ave:2}=0.5$, therefore it is true for n=2.

Inductive assumption: Assume that it is true for n=k. That is, $I_{ave:k}=\frac{k(k-1)}{4}$.

Inductive proof: $I_{ave:k+1}=\frac{(k+1)(k+1-1)}{4}=\frac{k(k+1)}{4}$. So it is true for n=k+1.

Therefore the statement is true for size n≥2 of a list A_n.

References

https://www.youtube.com/watch?v=ruBnYcLzVlU

Textbook pg 38-44, 2.2

2. The conventional one (k=2), the sorting time is $T_2(n) = cnlog_2n$.

T(n) = 2T(n/2) + n Substituting T(n/2) = 2T(n/4) + n/2

T(n) = 2T(n/2) + n = 2[2T(n/4) + n/2] + n = 4T(n/4) + n + n

T(n) = 4T(n/4) + 2n, Substituting T(n/4) = 2T(n/8) + n/2

T(n) = 4T(n/4) + 2n = 4[2T(n/8) + n/4] + 2n = 8T(n/8) + n + 2n

T(n) = 8T(n/8) + 3n Substituting T(n/8) = 2T(n/16) + n/2

T(n) = 8T(n/8) + 3n = 8[2T(n/16) + n/8] + 3n = 16T(n/16) + n + 3n

T(n) = 16T(n/16) + 4n

Rewrite the above equations and look for a pattern.

1st step of recursion	T(n) = 2T(n/2) + n	$= 2^1T(n/2^1) + 1n$
2nd step of recursion	T(n) = 4T(n/4) + 2n	$= 2^2T(n/2^2) + 2n$
3rd step of recursion	T(n) = 8T(n/8) + 3n	$= 2^3T(n/2^3) + 3n$

4th step of recursion $T(n) = 16T(n/16) + 4n$ $= 2^4T(n/2^4) + 4n$

Closed-form formula

$T(n) = 2^kT(n/2^k) + kn$.

We want T(1), (here, note that the T(1) condition isn't the same as in our assignment, but doesn't change it much) so we let $n=2^k$. Solving for k, we get k=logn. Now plug back in.

$T(n) = 2^{logn}T(2^k/2^k) + (logn)n = n*T(1) + (logn)n = n + nlogn$

$T(n) = n + nlogn$.

Modified mergesort

If we substitute any k>2 into the equations above, the closed-form fomula will be:

when k=3 $3^kT(n/3^k) + kn$

when k=4 $4^kT(n/4^k) + kn$

when k=5 $5^kT(n/5^k) + kn$

when k=6 $6^kT(n/6^k) + kn$

Thus the sorting time will be $T_k(n) = cnlog_kn$, hence $T_k(n) = cnlog_kn$.

This means that the greater the value of k, the faster the sorting time.

Therefore the modified mergesort could be faster for some k>2 than the conventional one (k=2).

Reference

https://www.facebook.com/photo.php?fbid=796003913784439&set=gm.1454563134829522&type=1&theater

Textbook pg 45-48, 2.3

3. $T_{qselect}(n) = cn$

$T_{qsort}(n) = cnlog_2(n)$

$cn < cnlog_2(n)$

$1 < log_2(n)$

$2 < n$ Therefore option (a) is faster for any n greater than 2.

For k=15 and $n=100,000,000 \approx 2^{26.6}$, (while c is the same for both)

$T_{qselect}(n) = cn = 100000000c$

$T_{qsort}(n) = cnlog_2(n) = 2657542475.90989c$ $100000000c < 2657542475.90989c$

Therefore option (a) is faster.

Reference

Textbook pg 48-54, 2.4

4. Worst case - when the pivot is the max or min and the list is in a reverse order.

Average case - when the pivot is the median and the list is in a random order.

The quicksort will always eliminate the inversions between the pivot in its initial position and the other elements in the list. This is because all elements less than the pivot will be on the left and all elements greater than the pivot will be on the right, so that there will be no inversions between the pivot in its sorted position and other elements in the list.

Reference

Textbook pg 48-54, 2.4

5. Process(a) - in linearithmic time O(nlogn), heapsort repeats n times the deletion of the maximum key and restoration of the heap property, where each restoration is logarithmic in all best, worst, and average case).

Process(b) - building a heap in linear time Θ(n) is a recursive structure. This means that the comparing process starts from the base case. The lower bound is Θ(n) since every input element must be inspected.

Because process(a) does not have the recursive structure, and also that the heapsort in process(a) repeats n times the deletion of the maximum key and restoration of the heap property, process(b) yields such acceleration.

Reference

Lecture11 slide 21-23

Textbook pg 55-62, 2.5