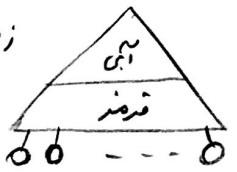
(2) اگر آرایه sort شده باشد، بدترین حالت است. تبدیل آن به heap $O(n)$ زمان می‌برد؛ سپس تابع Max-heapify فراخوانی شده که $\lg n$ عملیات انجام می‌دهد:

$$\sum_{i=1}^{n} \lg i \geq \sum_{\lceil n/2 \rceil}^{n} \lg i \geq \sum_{\lceil n/2 \rceil}^{n} \lg n/2 \geq \frac{n}{2} \lg \frac{n}{2} = \frac{n}{2}(\lg n - \lg 2) = \frac{n}{2}\lg n - \frac{n}{2}$$ (I)

$$\lim_{n\to\infty} \frac{n/2}{n/2 \lg n} = 0 \Rightarrow \frac{n}{2}\lg n - \frac{n}{2} = \Theta(n \lg n) \Rightarrow \lg n! = \Omega(n \lg n)$$ (I)

---

(2) فرض کنیم که heap یک درخت باینری کامل است ($n = 2^k - 1$)، پس 2^{k-1} برگ و $2^{k-1} - 1$ گره داخلی دارد. اگر این زیردرخت را در نظر بگیریم، حداکثر 2^{k-2} گره قرمز داریم؛ پس حداقل $2^{k-2} - 1$ گره آبی داریم. در صورت فراخوانی تابع heapify، گره‌های قرمز می‌توانند مستقیماً به ریشه بروند در حالی که گره‌های آبی باید swap شوند (کمترین تعداد swap وقتی است که $2^{k-2} - 1$ گره آبی به شکل درخت باینری داریم.) فرض کنید d گره آبی داشته باشیم؛ پس زیردرخت مذکور $\lg d$ سطح دارد که هر سطح i $2^i \times i$ گره دارد. تعداد swap‌ها:



$$\sum_{i=0}^{\lg d} 2^i \times i = 2\left(2^{\lg d} \lg d - 2^{\lg d} + 1\right) = 2d \lg d - 2d + 2 = \Omega(d \lg d)$$

زیردرخت مذکور نصف heap است؛ پس به صورت بازگشتی داریم: $T(n) = T(n/2) + \Omega(n \lg n)$

حالت سوم $\xrightarrow{\text{قضیه اصلی}}$ $T(n) = \Omega(n \lg n)$

---

(3) از هر لیست یک عضو خارج کرده و در یک min-heap (به صورت زوج مرتب (tuple) دوتایی که عنصر دوم، شماره‌ی لیستی که عضو مذکور از آن آمده را مشخص می‌کند) قرار می‌دهیم. هنگام merge، در هر مرحله کوچکترین عضو heap را پیدا کرده و بقیه‌ی عناصر آن لیستی که این عضو از آن آمده است را در heap قرار می‌دهیم. سپس از heap درون لیست اصلی قرار می‌دهیم. n ورودی داریم که قرار دادن هر کدام در heap، $\lg k$ طول می‌کشد؛ پس: $O(n \lg k)$

---

(4) از آن جا که همه‌ی ورودی‌ها برابرند، تابع partition آخرین خانه را برمی‌گرداند؛ پس:

$$T(n) = T(n-1) + n \leq c_1(n-1)^2 + c_2 n = c_1 n^2 - 2c_1 n + c_1 + c_2 n \leq c_1 n^2$$
$$(2c_1 > c_2 \,,\, n \geq \frac{c_1}{2c_1 - c_2})$$

$$\Rightarrow T(n) = T(n-1) + n = \Theta(n^2)$$

(5) در هر مرحله، تابع partition کوچکترین خانه را برمی گرداند؛ پس (مانند سوال قبل):

$$T(n) = T(n-1) + n = \Theta(n^2)$$

---

(6) برای مجموعه $\{a_1, a_2, \dots, a_n\}$ که به صورت $a_1 \leq a_2 \leq \dots \leq a_n$ هستند، $n-1$ ترتیب نسبی وجود دارد؛ پس کمترین عمق برگ: $n-1$

---

(7) ابتدا همه ی اعضا به مبنای n تبدیل می شوند (اعداد بین 0 تا n^2-1 هستند پس حداکثر $\lceil \log_n(n^2-1) \rceil = 2$ دو رقمی می شوند). در هر مرحله Sort برای هر رقم n حالت وجود دارد. زمان radix Sort $= O(d(n+b))$

$$\Longrightarrow O(2(n+n)) = O(4n) = O(n)$$

---

(8) بدترین حالت وقتی اتفاق می افتد که همه ی ورودی ها در یک bucket قرار بگیرند که $O(n^2)$ می شود.

bucketsort استفاده شوند Quicksort، Heapsort و Mergesort می توانند برای بهبود بدترین زمان بدون اینکه حالت میانگین را تغییر دهند.