RADIX SORT

Counting sort runs in $\Theta(n+k)$ where n is the number of elements to sort and $k = \max_{x\in n}(x)$. Therefore if you make d passes total running time is $\Theta(d(n+k))$. Suppose given n each b bits divided into r bit groups. Each of the $d = \lceil b/r \rceil$ groups and so radix sort runs in

$$\Theta\left(\frac{b}{r}(n+2^r)\right)$$

since for r bits the max number is 2^r.

For given values of n and b what's the best grouping? What's the best $r \le b$ that minimizes $(b/r)(n+2^r)$? If $b < \lfloor \lg n \rfloor$ then for any values of $r \le b$ we have that

$$\begin{aligned}
\Theta(n+2^r) &\le \Theta\left(n+2^{\lg n}\right) \\
&= \Theta(n+n) \\
&= \Theta(n)
\end{aligned}$$

Hence choosing $r = b$ we get

$$\Theta\left(\frac{b}{r}(n+2^r)\right) = \Theta(n)$$

If $b \ge \lfloor \lg n \rfloor$ choosing $r = \lfloor \lg n \rfloor$ gives the best running time within a constant factor. Why? With $r = \lfloor \lg n \rfloor$ we have

$$\Theta\left(\frac{b}{r}(n+2^r)\right) = \Theta\left(\frac{bn}{\lg n}\right)$$

If $r > \lfloor \lg n \rfloor$ then the 2^r term in the numerator dominates the r in the denominator and so is greater, i.e. the running time is bounded below:

$$\Omega\left(\frac{bn}{\lg n}\right)$$

If instead $r < \lfloor \lg n \rfloor$ then b/r increases and $n+2^r$ stays $\Theta(n)$.

Exercise 1 (8.3-4)**.** In base n the number n^3-1 are two digits numbers e.g. $1000_n = 1\times n^3 + 0n^2 + 0n + 0\times 1$. So we make 4 passes using radix sort

$$\Theta(4(n+n)) = O(n)$$

Exercise 2 (8-5)**.**

a. To be sorted duh
b. $2, 1, 4, 3, 6, 5, 8, 7, 10, 9$
c. Think about it: in

$$\sum_{j=i}^{i+k-1} A[i], \sum_{j=i+1}^{i+k} A[i]$$

all the "middle" elements are in common, i.e.

$$\begin{aligned}\sum_{j=i}^{i+k-1} A[i] &= A[i] + \sum_{j=i+1}^{i+k-1} A[i] \\ \sum_{j=i+1}^{i+k} A[i] &= \sum_{j=i+1}^{i+k-1} A[i] + A[i+k]\end{aligned}$$

Hence

$$\sum_{j=i+1}^{i+k} A[i] - \sum_{j=i}^{i+k-1} A[i] = A[i+k] - A[i] \geq 0$$

iff $A[i] \leq A[i+k]$ for $i = 1, \ldots, n-k$.

d. Effect the above: sort $A[1], A[1+k], \ldots$ and $A[2], A[2+k], \ldots$ and etc. How many entries for each of these sorts? n/k. So each sort costs $O((n/k)\log(n/k))$. How many sorts? k. So total cost is $O((k \times n/k)(\log(n/k))) = O(n \log(n/k))$.