**Problem 4-2**

a.1. Binary search, pass-by-pointer: $T(n) = T\left(\frac{n}{2}\right) + c$ for some $c$. In terms of $n$ (current problem size), this is $\Theta(\lg n)$, and in terms of $N$ (original problem size), this is $\Theta(\lg N)$.

a.2. Binary search, pass-by-copy whole array: $T(n) = T\left(\frac{n}{2}\right) + cN$ for some $c$. In terms of $n$, the term $cN$ acts as a constant, and this is $\Theta(\lg n)$, with $N$ as a hidden constant factor. But in terms of $N$, this becomes $\Theta(N \lg N)$

a.3. Binary search, pass-by-copy subrange: $T(n) = T\left(\frac{n}{2}\right) + c\frac{n}{2}$ for some $c$ (or, we could substitute $d = \frac{c}{2}$ to make $dn$). Using the master method, this is $\Theta(n)$, but in terms of $N$, this becomes $\Theta(N)$.

**Problem 4-2**, continued

b.1. Merge-sort, pass-by-pointer: $T(n) = 2T\left(\frac{n}{2}\right) + cn$ for some $c$. (To be even more precise, we could add a constant term $d$.) In terms of $n$, this is $\Theta(n \lg n)$, which becomes $\Theta(N \lg N)$ in terms of $N$.
b.2. Merge-sort, pass-by-copy whole array: $T(n) = 2T\left(\frac{n}{2}\right) + cn + dN$. Here we can use the recursion-tree method to observe the effect of the $dN$ term. Adding these terms over levels in the tree gives us

$$dN + 2dN + 4dN + \cdots NdN = dN \sum_{i=0}^{\lg N} 2^i \approx dN 2^{\lg N} = dN^2$$

And hence, in terms of $N$, this is $\Theta(N^2)$.
b.3. Merge-sort, pass-by-copy subrange: $T(n) = 2T\left(\frac{n}{2}\right) + cn$, which is $\Theta(n \lg n)$ in terms of $n$, or $\Theta(N \lg N)$ in terms of $N$.

**Problem 4-5**

**a.** Enumerate all pairings, of which there are $n!$. Consider a directed graph representing each node as a chip and the "good" results as edges. The chips that are good will be a strongly connected component. If more than $\frac{n}{2}$ are good, then the majority will be that strongly connected component, and so the largest strongly connected component will take up most of the chips. However, if less than $\frac{n}{2}$ are good, then the bad could mimic the good and form a strongly connected component just as big.

**b.** Pair them up and test them. That will be $\lfloor \frac{n}{2} \rfloor$ halves. What we want to do is keep a portion of them so as to retain the condition that a majority are good. When a pair indicates each that the other is good, then either they are both good or both bad. We'll keep one of them (arbitrarily). Claim: If there are an even number of chips and majority good, then the ones we keep will have a majority good.

**Proof.** *Suppose there are an even number of chips with a majority good, and we pair-up and eliminate as described.*

*Let $a$ be the number of pairs of both (actually) good chips, $b$ the number that are a mix of good and bad, and $c$ the number that are both (actually) bad. Note that there are $2a + 2b + 2c$ chips in total, $2a + b$ good ones, and $b + 2c$ bad ones. there is a majority good ones, $2a + b > b + 2c$, so $a > c$.*

*In a good/bad mixed pair, the good one will identify the bad one as bad, so the number of pairs reporting that they're both good is no more than $a + c$. (That maximum would happen if in all the bad-bad pairs the bad chips lie for each other.)*

*a good chips will be kept and* **at most** *$c$ chips will be kept. Since $a > c$, the chips kept are majority good.* □

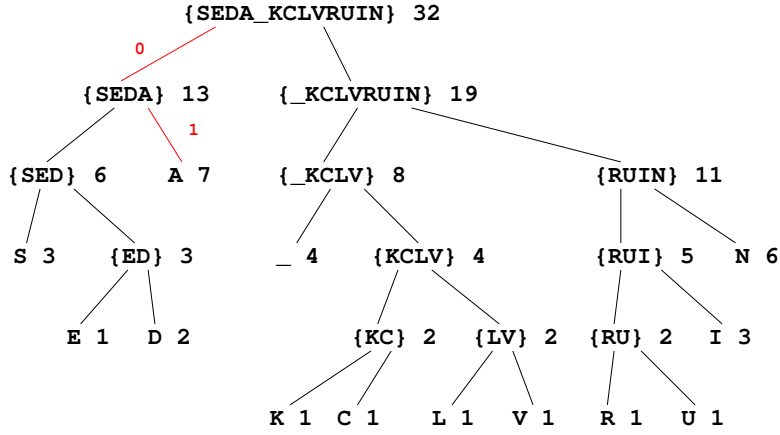
But what about the spare chip in the case of an odd number? This makes a difference because it would be possible for that spare chip to be the good one that gives the good majority:

> *Suppose we discard the spare chip. Suppose further that the chips are paired as BB, GG, G, and the BB pair reports as GG. Then we keep one B and one G, and we lose the G majority.*
> *Suppose we keep the spare chip. Suppose further that the chips are paired as GG, GG, BB, B, and the BB pair reports as GG. Then we keep two Gs and two Bs, and we lose the G majority.*

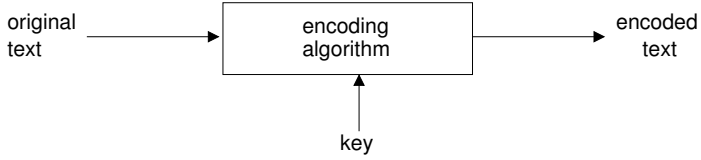
**My solution.** Have the other surviving chips vote. The other surviving chips have at least 50% good (as shown in the previous paragraph). If the spare chip gets at least 50% of the vote, it's good too. This exceeds $\lfloor \frac{n}{2} \rfloor$ comparisons, but is still linear.

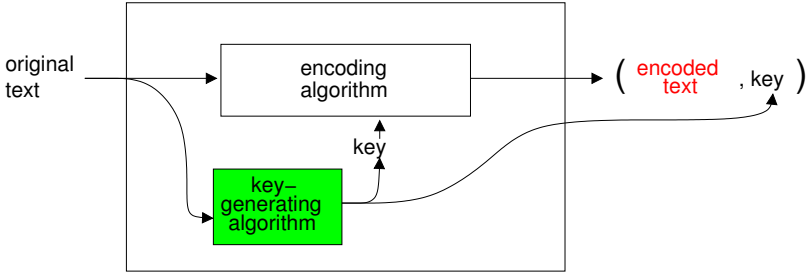
**Student solution.** If there are an even number of surviving chips, keep the spare. If there an an odd number of surviving chips, discard the spare.

From DMFP:




| 01 | 111 | 111 | 1101 | 10100 | 01 |
|---|---|---|---|---|---|
| A | N | N | I | K | A |


original
text
encoding
algorithm
encoded
text
key


original
text
encoding
algorithm
( encoded
text
, key )
key
key–
generating
algorithm

**Lemma 16.2,** restated from CLRS pg 433:
Let $x$ and $y$ be characters in n alphabet with the lowest frequencies in the original text. Then there exists an optimal prefix code for the alphabet (that is, *optimal for the original text*) in which which the encodings of $x$ and $y$ have the greatest length.
**Proof sketch.** Let $T$ be an optimal tree. Let $a$ and $b$ be characters represented by sibling leaves of maximal depth. WOLOG, let $a.freq \leq b.freq$ and $x.freq \leq y.freq$. By how $x$, $y$, $a$, and $b$ are chosen,

$$\begin{aligned} x.freq &\leq a.freq \\ \\ y.freq &\leq b.freq \end{aligned}$$

Let $T''$ be the prefix code like $T$ except with $x$ and $a$ switched, $y$ and $b$ switched. Then

$$\begin{aligned} B(T) - B(T'') &= \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T''}(c) \\ \\ &= (a.freq - x.freq)(d_T(a) - d_T(x)) \\ &\quad +(b.freq - y.freq)(d_T(b) - d_T(y)) \\ \\ &\geq 0 \end{aligned}$$

□

**Lemma 16.3,** summarized from CLRS pg 435:
Optimal trees have subtrees that are optimal for their corresponding subproblem.

**Theorem 16.4,** restated from CLRS pg 435:
Huffman trees are prefix codes that are optimal for the given original text.

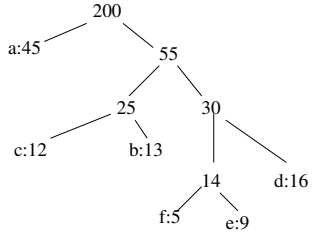
**Proof sketch.** Let $C$ be the alphabet of the text, augmented with character frequencies. By induction on the structure of the tree produce by the Huffman encoding.

**Base case.** Suppose $C$ has only one character. Then there is only one possible tree for that alphabet, so it must be optimal.

**Inductive case.** Suppose $C$ has more than one character, and let $x$ and $y$ be the the least frequent characters. Let $C'$ be the alphabet like $C$ but with $x$ and $y$ replaced with pseudo-character $z$. By structural induction, the Huffman encoding produces a tree that is optimal for $C'$. By Lemma 16.3, we can replace leaf $z$ in the optimal tree for $C'$ with a parent of siblings $x$ and $y$ to make a tree optimal for $C$. $\square$

**Solution to 16.3-2.** Suppose $T$ is a non-full prefix code binary tree. Let $x$ be a node with one child. Replace that node with its child; that reduces the depth of all characters underneath by 1. Hence $T$ was not optimal.

**Solutiuon to 16.3-4.** Claim: sum of the internal nodes' combined frequencies equals sum of the products of leaf frequencies and their depths. For example, consider this tree:




In this case, e's 9 occurrences each take 4 bits. The 9 is counted four times.
For an internal node $x$, the sum of the internal nodes' combined frequency of children is equal to the sum of leaf frequencies times their depth from $x$.

**Solutiuon to 16.3-4, continued.**

**Proof.** By structural induction.

Base case: Suppose $x$ is an internal node both of whose children, $a$ and $b$, are leaves. Then the combined frequency is

$$a.freq + b.freq = a.freq \cdot 1 + b.freq \cdot 1$$

. . . which is the leaf frequencies times their depths.

Inductive case 1: Suppose $x$ is an internal node with one child being a leaf ($a$) and the other being itself an internal node ($c$); suppose that the claim we're making for the entire tree is true for the subtree rooted at $c$. Let $d$ be the combined frequency of $c$ and $d'$ the sum of the combined frequencies of internal nodes under $c$. Let $c_1, \ldots c_m$ be the leaves under $c$ with depths (from $c$), $c'_1, \ldots c'_m$. Then the sum of the combined frequencies under $x$ is

$$\begin{aligned} a.freq + d + d' &= a.freq + d + c'_1 \cdot c_1.freq + \ldots c'_m \cdot c_m.freq && \text{by the ind hyp} \\ &= a.freq + c_1 + \cdot + c_m + \\ &\quad + c'_1 \cdot c_1.freq + \ldots + c'_m \cdot c_m.freq \\ &= 1 \cdot a.freq + (c'_1 + 1)c_1.freq + \ldots (c'_m + 1)c_m.freq \end{aligned}$$

The argument is similar in inductive case 2, where both children are themselves internal nodes. □