1 Problem 2

The following examples shows how the integers are stored exactly in R.

$2 = (-1)^0 * 1.0 * 2^{1024-1023}$

$-3 = (-1)^1 * (2^1 + 2^0) = (-1)^1 * (1 + 2^{-1}) * 2^{1024-1023}$

$2^{53}-1 = 2^52+2^51+...+2^1+2^0 = (-1)^0*(1+2^{-1}+2^{-2}+...+2^{-52})*2^{1075-1023}$

We then consider the situation when integers are larger than $2^{53} - 1$. The results below shows that $2^{53}, 2^{53} + 2$ can be represented exactly but $2^{53} - 1$ cannot.So we claim that the spacing of numbers of this magnitude is 2.

```
a <- 2^53-1
b <- 2^53
c <- 2^53 + 1
d <- 2^53 + 2

#find out how a,b,c,d are stored
identical(b-a,1)

## [1] TRUE

identical(c-b,1)

## [1] FALSE

identical(d-b,2)

## [1] TRUE
```

This is because

$2^{53} = (-1)^0 * (1.0) * 2^{1076-1023}$

$2^{53} + 2 = (-1)^0 * (1 + 2^{-52}) * 2^{1076-1023}$

They can be repremted exactly.

$2^{53} + 1 = (-1)^0 * (1 + 2^{-53}) * 2^{1076-1023}$

2^{-54} is less than 2^{-53} the minimum number that the computer can store. So it can not be stored exactly.

For numbers staring with 2^{54}

$2^{54} = (-1)^0 * (1.0) * 2^{1077-1023}$

$2^{54} + 1 = (-1)^0 * (1 + 2^{-54}) * 2^{1077-1023}$

$2^{54} + 2 = (-1)^0 * (1 + 2^{-53}) * 2^{1077-1023}$

$2^{54} + 3 = (-1)^0 * (1 + 2^{-53} + 2^{-54}) * 2^{1077-1023}$

$2^{54} + 4 = (-1)^0 * (1 + 2^{-52}) * 2^{1077-1023}$

Since 2^{-53},2^{-54} are smaller than 2^{-53}, we can't precisely represent $2^{54} + 1$,$2^{54} + 2$, $2^{54} + 3$. So the spacing of numbers of this magnitude is 4.

2 Problem 3

2.1 Problem a

The following results show that even if it takes longer to create a numeric vector, it doesn't imply it takes longer to copy a numeric vector. Note that if we directly copy a vector using syntax like "a¡-integervec", no actual copy will be made.

```
p <- 1e7
vector1 <- vector()
vector2 <- vector()



system.time(integervec <- 1:p)
##user system elapsed
  0.02   0.03    0.04

system.time(numericvec <- rnorm(p))
##user system elapsed
  3.76   0.00    3.78

system.time(vector1 <- c(vector1,integervec))
##user system elapsed
  0.08   0.03    0.11

system.time(vector2 <- c(vector2,numericvec))
##user system elapsed
  0.06   0.06    0.12
```

2.2 Problem b

The following results show that it is not necessarily faster to take a subset of size roughly n/2 from an integer vector of size n than from a numerica vector of size n.

```
#This tests the time difference when taking the first half of each vector.

system.time(integervec[1:round(p/2)])
##user system elapsed
  0.15   0.03    0.19

system.time(numericvec[1:round(p/2)])
##user system elapsed
  0.13   0.06    0.19
```

```r
#This tests the time difference when taking the vector elements with odd indices.
system.time(integervec[c(TRUE, FALSE)])
##user system elapsed
  0.98   0.02    1.01

system.time(numericvec[c(TRUE, FALSE)])
##user system elapsed
  0.92   0.04    0.99
```

3 Problem 4

3.1 Problem a

I think there are several reasons for this.

First, if we break up Y into n columns, it's possible that the overhead of threading outweights the gains from distributing the computations.

Second, it can avoid the possible problem of adding or subtracting numbers that are very different in magnitude, i.e. the partial sum may be much larger than the new term if we add the X*Yi one by one. Breaking up Y into p blocks is kind of like doing the summation in a tree like fashion.

3.2 Problem b

Amount of memeory used wehn all p workers are doing their calculations:

Approach A:$p * (n * n + m * n + n * m) = (1 + 2/p)n^2$

Approach B:$p * (m * n + m * n + m * m) = (1/p^2 + 2/p) * n^2$

Communication cost:

Total number of numbers passed to the worker:

Approach A:$p * (n * n + n * (n/p)) = (p + 1) * n^2$

Approach B:$p^2 * (n * m + n * m) = 2 * p * n^2$

Total number of numbers passed to the master:

Approach A:$p * n * (n/p) = n^2$

Approach B:$p^2 * (n/p)^2 = n^2$

Total communication cost:

Approach A:$(p + 1) * n^2 + n^2 = (p + 2)n^2$

Approach B:$2 * p * n^2 + n^2 = (2p + 1) * n^2$