The tipping point of big data: a quantitative definition of heavy data derived from economics and data analytics.

© April 2015 Andrew C Lea, MA (Cantab), FBCS
Primary Key Associates

This paper argues that data passes the 'tipping point of big data' when the cost of analysing it becomes greater than the value of that analysis, and that this is in turn governed by the running time or Order of the algorithms we need to employ. When data passes the tipping point of big data, it becomes heavy data: quantitatively different.

By measuring current costs, benefits, and data sizes, and by understanding the Order of the analytical functions employed, the tipping point at which data becomes heavy - too big to be financially worth analysing - the tipping point of big data - can be calculated.

Furthermore, by considering likely growth of data sizes and processing power, the time at which that tipping point will be reached can be estimated. We discover that for some algorithms the cost of processing will render the analysis of heavy data financially non-viable within a matter of years. Efficient algorithms will become more important, not less, because of the increase in processing power.

1 Introduction

1.1 Objectives

From the earliest days of computing, we have always had 'too much' data, and this has provided a strong motivation in the development of computers. Arguably one of the first uses of mechanical data analysis - the US 1890 census, processed using Hollerith punched cards[1] - was an instance of big data.

The concept of big data appeals - possibly because of the marketing opportunities it provides - but the "velocity, volume, and variety" definitions of big data are largely qualitative. The current definitions do not allow us to address the question of "will this quite-big data ever become too-big, if so when, and what can be done about it?".

Because big data is *quantitatively* ill-defined, other terms are being coined, such as "Really big data" [2]. 'Big data' also includes the concept of 'Big Data in motion" [3], also known as 'velocity'.

In this paper we will seek to define ***heavy data*** - which is big data that has passed the tipping point of big data – by means of an *economic* definition of the tipping point of big data. We will define that point, the point at which data becomes too big to be economically worth analysing, based on the:

- economic value of the insights gained through analysing the data;
- efficiency of the algorithms analysing that data; and the

- cost of performing that analysis.

This is, of course, primarily a volume based definition. It encompasses velocity in so far as velocity (of data) is volume over time. Variety is subsumed in so far as it is often the variety which causes the complexity of analysis.

1.2 Candidate Big Data and Heavy Data

We distinguish between big collections of small data, and big data. Data which is a large collection of small data sets, each of which can be valuably analysed independently of the other, is not big data. Big data has potential dependencies throughout the data set, and cannot therefore be analysed sub-set at a time. This does not necessarily mean that big data cannot be an aggregation of smaller data sets, but it does imply that the analysis to be conducted ranges across all those fused data sets.

Heavy data is big data which is too expensive to analyse, and for any particular context, inherits the qualitative definitions of big data, and the quantitative definition of the tipping point of big data.

1.3 Analysis of Algorithms

In this paper we primarily consider algorithms which are intended to extract distilled meaning from datasets, often referred to as data mining. An application area we will refer to throughout this paper is that of fraud discovery, in which the signal to noise ratio is, hopefully, low (as otherwise we are experiencing heavy levels of fraud). It is a common feature of data mining algorithms that the intention is to reduce high volumes of (frequently poor quality) data to actionable information. These algorithms have to range across the entire dataset.

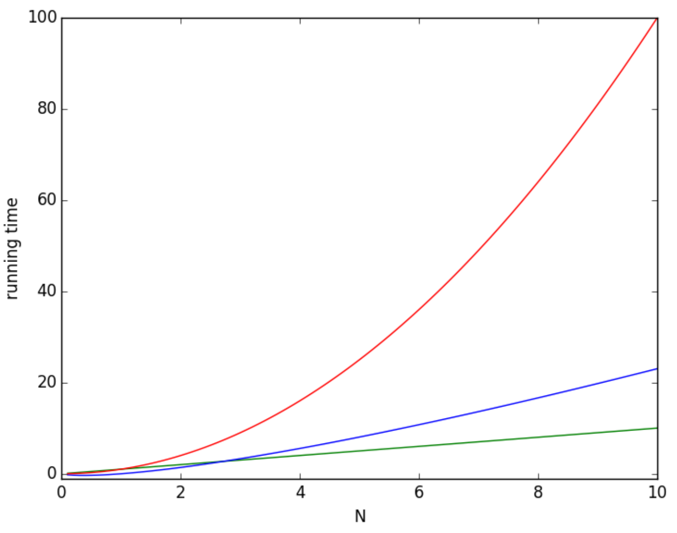
The formulas developed here are primarily developed for, and are therefore most appropriate to, processing algorithms, as opposed to those, say, concerned with storage, indexing, and data retrieval.

For the analysis of algorithms in a **single processing** environment we will use the Order notation [4], , which essentially uses the most significant power of the running time as its characteristic:

The very fastest algorithms grow with *Order(N)*: in other words a data set ten times bigger than another takes ten times as long to analyse. Unfortunately, it seems that only the simplest algorithms (such as calculating means and variance) grow at this rate. It is difficult to see how an algorithm could run faster than *Order(N)*, since it would imply that not all of the N data items are visited even once. Meta-data can frequently be used to aid algorithms by, for example, grouping data. In this case the relevant size of the data set is not the atomic values, but the number of groups. This paper also excludes sub-linear[5] approaches, which are clearly very valuable in the right circumstances. These might include statistical and sampling approaches, quantum computing, and probabilistic 'algorithms'. We exclude them not because of lack of value, but because of the approximate solutions they find.

Fast search algorithms - such as binary search [6] - execute at *Order(N.log(N))*, and for many types of problems this is as good as optimised algorithms get. We shall use *Order(N.log(N))* as our realistic best case.

Many algorithms grow as *Order(N^2)*, such as all pairwise comparisons. A data set ten times bigger than another takes one hundred times as long to analyse. We shall take this as our realistic worst case. Slower and more powerful algorithms certainly exist, but we are unlikely to use them on big data, because of their prohibitive running time and costs.



These running times vs data set size N are shown on the right for *Order (N)*, *Order(N.log(N))*, and Order(N^2) in green, blue, and red respectively.

Whilst **parallel processing** may reduce analytical time, it does not decrease the cost of analytics, since the number of machine instructions (with the electrical and cooling costs) remains the same. In a shared CPU environment, parallel processing need not increase the cost compared to single-task analysis, since cores not needed are either used for other purposes or powered down.

Parallel processing can increase analysis cost if speed is of the essence. In favourable circumstances we may choose to run less efficient algorithms which, whilst using more instructions overall, result in quicker execution time because of their ability to execute parts of the algorithm in parallel. We do not need to consider such algorithms in this paper. Their analysis is complex, and the subject of much research in its own right[7].

Data storage costs do not define when data becomes big since, unlike algorithms, data storage costs grow at worst with *N*, and with economies of scale, probably less than *N*.

2 Full Derivation

In this section we derive equations for evaluating the size, *N*, at which a dataset passes the big data tipping point and becomes heavy data.

2.1 *Benefit of analysis*

In this section we refer to the unit of insight that is uncovered by analytics. For example, it might be the number of frauds detected, or the number of marketing opportunities identified. We analyse data, big or otherwise, because it provides some financial-equivalent benefit:

$$benefit = b.U(N)$$

where N is the size of the dataset, b is the financial value of each unit of insight, and U is a function returning the number of useful insights uncovered. (In some cases this may be a constant, independent of the number of insights gained. For example it may be the value attributed to a scientific experiment.)

As circumstances change, the value of the insight may change, changing the value of b; or the number of relevant insights may change, changing the form of U. In both cases, the tipping point of big data would change accordingly, but would still be governed by the equations developed here.

2.2 Cost of analysis

Analysing data costs electricity for powering the computer and cooling it:

$$cost = e.k.O(N)$$

where O is the equation returning the running time of the algorithm, k is a constant which maps that running time into processor seconds, and e is the cost of electricity ($/processor second). We can now write our $^{cost}/_{benefit}$ equation:

$$\frac{benefit}{cost} = \frac{b.U(N)}{e.k.O(N)}$$

We do not need to determine e, k, or b if we are already running our analytics, and wish to know when it will become too big. We substitute $v = {}^{b}/_{e.k}$, define $I(N) = {}^{O(N)}/_{U(N)}$ and measure both our total benefit (using management information systems) and total cost (using our accounting systems) and can write:

$$\frac{benefit}{cost} = \frac{v}{I(N)}$$

From this we can calculate v:

$$v = \frac{benefit.I(N)}{cost}$$

2.3 Tipping point of big data

Using our measured value of v, we can estimate the point at which data is exactly worth evaluating. In economists' terms, this is the point at which we are indifferent to whether or not analyse our data [8]. This is the point at which ${}^{benefit}/_{cost} = 1$:

$$\frac{benefit}{cost} = 1 = \frac{v}{I(N)}$$

If I'(y) is the inverse of I(x), then:

$$I'(N) = v$$

and the point of heavy data (within our algorithm and economic context) is therefore:

$$N = I'(v)$$

2.4 Example: Fraud Discovery with Order(N^2) algorithms

By way of example, a useful application of data mining on big data is the discovery of fraud. Typically the data which could be mined for fraud may comprise financial transactions, such as purchases, transaction history, customer details, credit card information, bank details, geographical data (where the transaction is occurring) and so forth. In many cases, frauds will look very similar to legitimate transactions. Let us consider purchases made with a stolen credit or debit card, and three approaches that might be used to detect the fraudulent transactions:

1. The thieves attempt to withdraw very large amounts at an ATM. In this case, a simple rule might be triggered as to the size of the transaction.
2. Car fuel is purchased using a credit card clone, in a transaction otherwise very similar to that frequently performed by the legitimate owner. In this case, a rule supplement by modest data mining, might indicate that the card is being used an implausibly long way from the legitimate card.
3. A series of subtle purchases are carried out using a cloned card, in a way designed to 'stay under the radar'. To detect these fraudulent transactions – which might be occurring in parallel with legitimate purchases – a pattern of use analysis might be needed, perhaps matching it to known fraudulent usages. This could require extensive data mining to find the pattern it is most similar too, and could well take the company into big data and – it it passes the tipping point – into heavy data mining.

We may suppose the number of frauds in a dataset to be roughly proportional to the square root of that dataset[9], in which case the **benefit of analysis** is given by:

$$benefit = b\,.\,f.\sqrt{N}$$

where b is the average saving made by discovering a fraud, and f is a constant of proportionality, relating the square root model to the number historically discovered. In other situations, a different relationship will hold.

Powerful algorithms to uncover fraud, such as comparing all transactions on a particular day with all other transactions (example 3 above), can have a **cost of analysis** as high as N^2:

$$cost = e\,.\,k.\,N^2$$

We can now write our **$^{cost}/_{benefit}$** equations:

$$\frac{benefit}{cost} = \frac{b.f.\sqrt{N}}{e.k.N^2} = \frac{b.f}{e.k.N^{3/2}}$$

We do not need to determine *b*, *f*, *e*, or *k* if we are already running our fraud detection, and wish to know when it will become too big. We substitute $v = {}^{b.f}/_{e.k}$ and measure both our total benefit and cost (using our management and accounting systems).

Let us assume the total value of frauds discovered is $50,000 per month, the cost of doing so is $5,000, and our dataset currently has 100,000,000 data points:

$$\frac{benefit}{cost} = \frac{v}{N^{3/2}} \quad \text{and so ...}$$

$$v = \frac{benefit . N^{3/2}}{cost} = 10^{12}.10 = 10^{13}$$

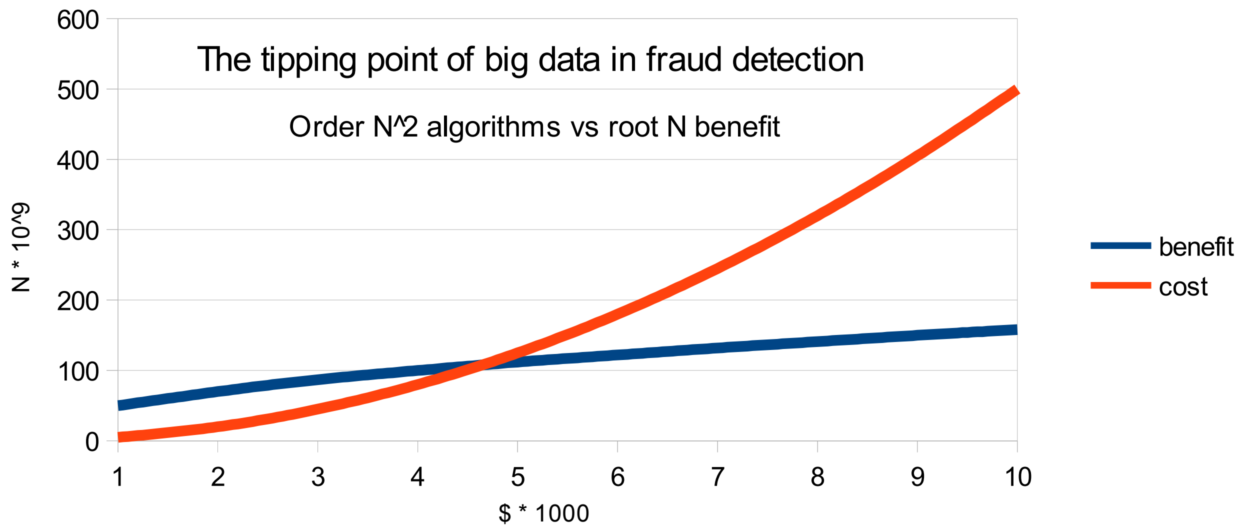
Using our measured value of v, we can estimate the point at which data is exactly worth analysing – the **tipping point** - when $^{\text{benefit}} / _{\text{cost}} = 1$:

$$\frac{benefit}{cost} = 1 = \frac{v}{N^{3/2}} \quad \text{and so ...}$$

$$N^{3/2} = v = 10^{13}$$

therefore the tipping point of big to heavy data, in our algorithm and economic context, is:

$$N = v^{2/3} = 10e13^{2/3} = 464{,}158{,}883$$



With this *Order(N*2*)* algorithm, should our data grow by only a factor of five, it will no longer be financially worth-while analysing the data to discover fraud. The tipping point of big data will have been passed, and our data will be too heavy to analyse.

3 Short Derivation

If costs and benefits are known, we can use a simpler analysis. The Order function of our analytics can be obtained in two ways:

1. from theoretical considerations, by analysis of the algorithms used
2. by measuring execution time with the current full data set, and smaller sub-samples of it, the growth of execution size vs data set size can be modelled.

3.1 Benefit of analysis

We analyse data, big or otherwise, because it provides some financial-equivalent benefit. This is necessarily so, since we chose to invest resource in undertaking the data analysis; if we considered that, given the means available to us, other methods would have a higher expected return or lower expected cost, we would have used them instead.

$$benefit = b.U(N)$$

where N is the size of the dataset, U is a function returning the number of useful insights uncovered, and b is the financial value of each unit ($/unit). We now solve for b:

$$b = \frac{benefit}{U(N)}$$

3.2 Cost of analysis

Similarly, analysing data costs electricity for powering the computer and for cooling it, staff to run it, and depreciation:

$$cost = c.O(N)$$

where O is the equation returning the running time of the algorithm, c is a constant which maps that running time into direct costs in $. Again, we can solve for c:

$$c = \frac{cost}{O(N)}$$

3.3 Tipping point of big data

The tipping point of big data is, as before, the point at which ${}^{benefit}/_{cost} = 1$. Using our measured values of b and c, we equate the equations of benefit and cost, and solve for N:

$$b.U(N) = c.O(N)$$

3.4 Example: Fraud Discovery with Order (N.log(N)) algorithms

Let us again assume the total value of frauds discovered is $50,000 per month, the cost of doing so is $5,000, and our dataset currently has 100,000,000 or 10^8 data points. In this case, however, we will use a more efficient heuristic with a running cost of *N.log(N)*.

We assume the same **benefit of analysis** as the previous example:

$$benefit = b\,.\,f.\sqrt{N}$$

$$b = \frac{benefit}{\sqrt{N}} = \frac{50{,}000}{\sqrt{100{,}000{,}000}} = \frac{50{,}000}{10{,}000} = 5$$

Algorithms need to uncover fraud can be as expensive as N^2, but lower power heuristics can easily have a **cost of analysis** of *N.log(N)*.

$$cost = c.N.log(N)$$

$$c = \frac{cost}{N.log(N)} = \frac{5{,}000}{10^8 . \log(10^8)} = \frac{5}{100{,}000 . 8} = \frac{1}{160{,}000}$$

We can now write our **cost / benefit** equation:

$$\frac{benefit}{cost} = \frac{b.sqrt\,N}{c.N.log(N)} = \frac{b}{c.sqrt\,N . \log(N)}$$

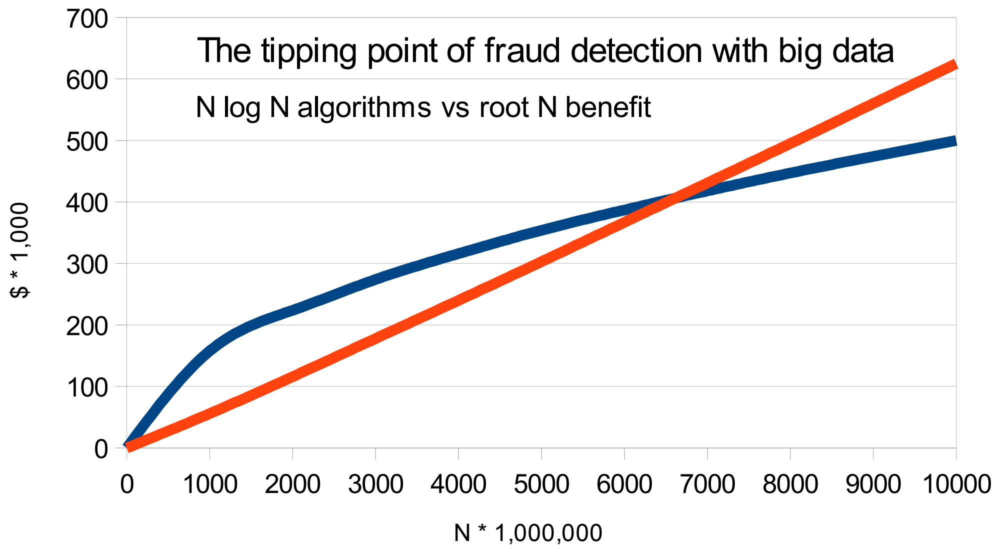
As, by our definition of **tipping point**, *benefit / cost = 1* and *benefit = cost*,

$$\frac{b}{c.sqrt\,N} . \log(N) = 1 \quad \text{and therefore} \quad \sqrt{N} . \log(N) = \frac{b}{c} = 5 . 160{,}000 = 800{,}000$$

so solving for N (in this case numerically) we find the tipping point of big to heavy data with this algorithm and context to be

N = 6,634,343,897

Alternatively, we could have simply plotted the cost and benefit vs *N*, and identified the intersection:



With this Order(N log(N)) algorithm, the tipping point of big data is not reached until our data has increased by a factor of more than 66.

4 Temporal Big Data and Heavy Data

In the previous section we saw how to calculate the tipping point of big data, assuming today's costs and benefits. Of great interest to those performing big data analysis is when that tipping point will be reached. Clearly this will depend on both the growth of the data to be analysed, and the growth in our processing power.

4.1 Benefit of analysis

The growth of our processing power (per $) and the growth of in the data to be analysed need not, of course, be governed by the same functions. Suppose *N* grows with time:

$$N_t = N_0.F(t)$$

$$benefit_t = b.U(N_0.F(t))$$

We solve for b as before, using N_0:

$$b = \frac{benefit}{U(N_0)}$$

4.2 Cost of analysis

The cost of analysis depends firstly on both the size of the data, and also on how our processing costs drop, which drop with *G(t)*:

$$cost_t = \frac{c.O(N_0 F(t))}{G(t)}$$

4.3 Tipping Point

As before, our tipping point arrives when cost = benefit. If costs grow less than benefit, there may be no tipping point.

$$b.U(N_0 F(t)) = \frac{c.O(N_0 F(t))}{G(t)}$$

4.4 Example: Moore's Law data growth vs Order N^2 algorithms

Let us now examine when the tipping point of big data will arrive for marketing analysis. Specifically, we assume:

- our initial dataset has 100,000,000 or 10^8 data points
- the benefit we derive is proportional to the number of customers, that our customer base grows at 10% per annum, and our initial benefit is $50,000 per month;
- the amount of data each customer generates grows with Moore's law;
- our cost of processing declines with Moore's law, and is initially $5,000 per month.

"Moore's Law" is the observation made (and revised) by Gordon Moore of Intel that processing power will approximately double every two years[10]. The exact period of doubling is of course greatly discussed, and the equations below can be readily adjusted to other periods if preferred. Processing capability (both in data generation and analysis) vs time t in years is given by:

$$P_t = P_0 2^{t/2}$$

The number of customers S vs time is given by:

$$S_t = S_0 1.1^t$$

The amount of data we shall have to process is therefore:

$$N_t = N_0 P_t S_t = N_0 1.1^t 2^{t/2}$$

We assume the same **benefit of analysis** as the previous example:

$$benefit_t = b\, S_0 1.1^t = 50{,}000\, 1.1^t$$

$$b = benefit_0 = 50{,}000$$

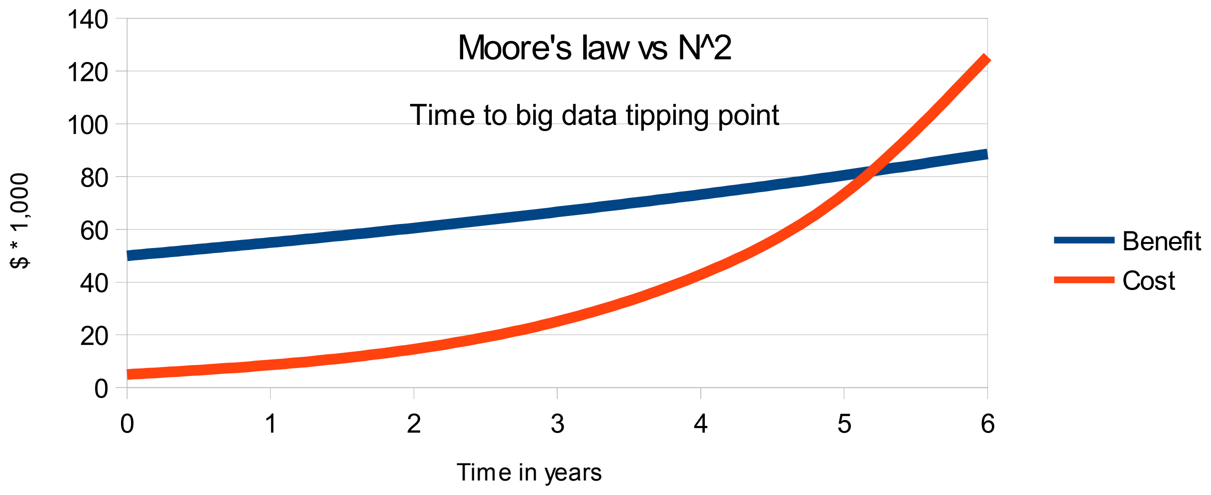
The **cost of analysis**

$$cost_t = c \frac{(N_0 F(t))^2}{G(t)} = c \frac{(N_0 1.1^t . 2^{t/2})^2}{2^{t/2}}$$

$$cost_0 = c . N_0^2$$

$$c = \frac{cost}{N^2} = \frac{5000}{(10^8)^2} = \frac{5}{10^{13}}$$

As before we can find the **tipping point** by solving for t in our $^{cost}/_{benefit}$ equation, or by plotting the cost and benefit vs time, and looking for the intersection:



Our big data will reach the tipping point and become heavy data in just over five years.

4.5 Example: Moore's law data growth vs Order(N.log(N)) algorithms

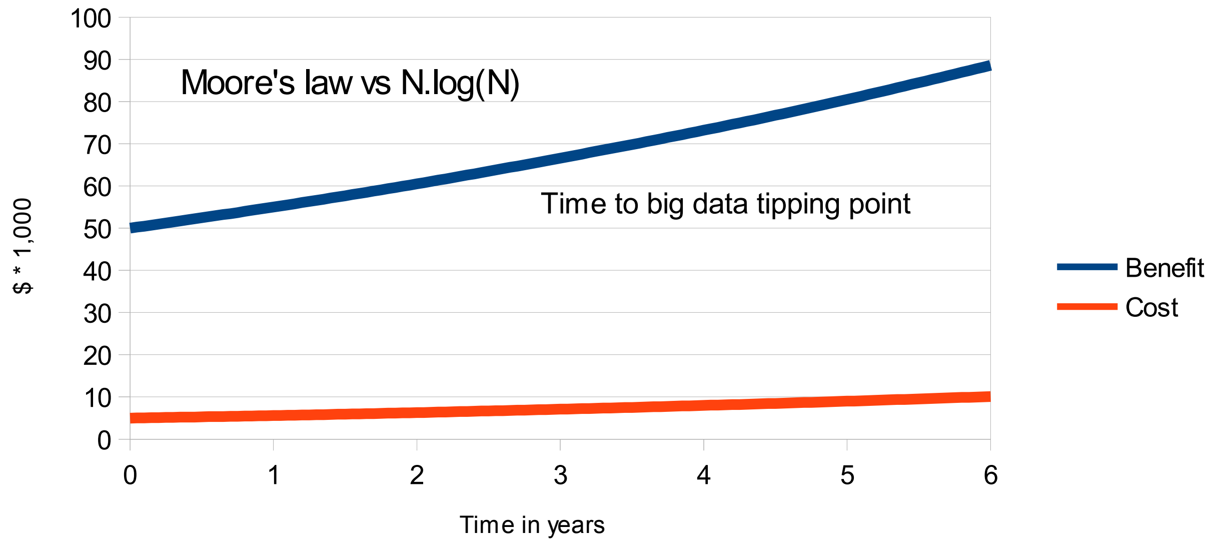
Repeating the same example, but against *Order(N.log(N))* algorithms, the cost and benefit constants remain the same as previously. The benefit equation remains unchanged too. Recall that the amount of data we have to process is:

$$N_t = N_0 1.1^t . 2^{t/2}$$

giving a **cost of analysis** of

$$cost = c\frac{N.log(N)}{G(t)} = c\frac{N_0 1.1^t 2^{t/2} \log(N_0 1.1^t 2^{t/2})}{2^{t/2}} = c.N_0 1.1^t . \log(N_0 1.1^t 2^{t/2})$$

Plotting the growth of both benefit and cost, we see that we never reach the **tipping point**:



5 Conclusions

Data is 'big' when:

- There are potential cross-dependencies throughout the data set, so that it is non-seperable, and cannot be partitioned into smaller sets for analysis

Data is 'heavy' (as well as big) as it crosses the tipping point of big data when:

- The cost of analysing it is greater than the benefit of so doing.

By knowing the Order running time of the analysing algorithm, the size of our current data set, and the current cost/benefit ratio, we can calculate at what size our data will reach the big data tipping point.

Both the running time of algorithms and how well they can be partitioned are, and will remain, crucial; not only regardless of, but because of, the growth in CPUs and CPU speed.

Efficient algorithms will become more, not less, important. Without efficient algorithms, the tipping point of big data – heavy data - will overwhelm many systems.

[1] Herman Hollerith, The Electronic Tabulating Machine, *Journal of the Royal Statistical Society*, Vol. 57, No. 4 (Dec., 1894), pp. 678-689

[2] J. B. Cole, S. Newman†, F. Foertter, I. Aguilar and M. Coffey: *BREEDING AND GENETICS SYMPOSIUM:* Really big data: Processing and analysis of very large data sets. Journal of Animal Science

[3] Hirzel, M. IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY, USA Andrade, H. ; Gedik, B. ; Jacques-Silva, G. ; Khandekar, R. ; Kumar, V. ; Mendell, M. ; Nasgaard, H. ; Schneider, S. ; Soule, R. ; Wu, K.-L. *IBM Streams Processing Language: Analyzing Big Data in motion*. IBM Journal of Research and Development (Volume:57 , Issue: 3/4) May-July 2013

[4] Donald Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms, Third Edition*. Addison–Wesley, 1997. ISBN 0-201-89683-4.

[5] Kumar, Ravi; Rubinfeld, Ronitt (2003). "Sublinear time algorithms". *SIGACT News* 34 (4): 57–67.

[6] Knuth, Donald (1997). "Section 6.2.1: Searching an Ordered Table". Sorting and Searching. *The Art of Computer Programming 3 (3rd ed.)*. Addison-Wesley. ISBN 0-201-89685-0.

[7] Journal of Parallel and Distributed Computing

[8] Steven Landsburg; *The Armchair Economist: Economics and Everyday Life*, 1993

[9] Obtained in discussion with a Certified Fraud Examiner.

[10] Moore, Gordon (2006). "Chapter 7: Moore's law at 40". In Brock, David. *Understanding Moore's Law: Four Decades of Innovation*. Chemical Heritage Foundation. pp. 67–84. ISBN 0-941901-41-6.