Truncated exponential backoff (https://en.wikipedia.org/wiki/Exponential_backoff) is a standard error handling strategy for network applications in which a client periodically retries a failed request with increasing delays between requests. Clients should use truncated exponential backoff for all requests to Cloud Storage that return HTTP `5xx` and `429` response codes, including uploads and downloads of data or metadata.

Understanding how truncated exponential backoff works is important if you are:

- Building client applications that use the Cloud Storage XML API (/storage/docs/xml-api/overview) or JSON API (/storage/docs/json_api/) directly.
- Accessing Cloud Storage through a client library (/storage/docs/reference/libraries). Note that some client libraries, such as the Cloud Storage Client Library for Node.js (/nodejs/docs/reference/storage/latest/), have built-in exponential backoff.
- Using the gsutil command line tool, which has configurable retry handling (/storage/docs/gsutil/addlhelp/RetryHandlingStrategy).

If you are using the Google Cloud Console (https://console.cloud.google.com/), the console sends requests to Cloud Storage on your behalf and will handle any necessary backoff.

An exponential backoff algorithm retries requests exponentially, increasing the waiting time between retries up to a maximum backoff time. An example is:

1. Make a request to Cloud Storage.
2. If the request fails, wait 1 + `random_number_milliseconds` seconds and retry the request.
3. If the request fails, wait 2 + `random_number_milliseconds` seconds and retry the request.
4. If the request fails, wait 4 + `random_number_milliseconds` seconds and retry the request.
5. And so on, up to a `maximum_backoff` time.
6. Continue waiting and retrying up to some maximum number of retries, but do not increase the wait period between retries.

where:

- The wait time is min(((2^`n`)+`random_number_milliseconds`), `maximum_backoff`), with `n` incremented by 1 for each iteration (request).
- `random_number_milliseconds` is a random number of milliseconds less than or equal to 1000. This helps to avoid cases where many clients get synchronized by some situation and all retry at once, sending requests in synchronized waves. The value of `random_number_milliseconds` is recalculated after each retry request.
- `maximum_backoff` is typically 32 or 64 seconds. The appropriate value depends on the use case.

It's okay to continue retrying once you reach the `maximum_backoff` time. Retries after this point do not need to continue increasing backoff time. For example, if a client uses an `maximum_backoff` time of 64 seconds, then after reaching this value, the client can retry every 64 seconds. At some point, clients should be prevented from retrying infinitely.

How long clients should wait between retries and how many times they should retry depends on your use case and network conditions. For example, mobile clients of an application may need to retry more times and for longer intervals when compared to desktop clients of the same application.

If the retry requests fail after exceeding the `maximum_backoff` plus any additional time allowed for retries, report or log an error using one of the methods listed under Support & help (/storage/docs/resources-support#support-packages).

Examples of truncated exponential backoff used with Cloud Storage include:

- A boto example (https://github.com/GoogleCloudPlatform/gsutil/blob/master/gslib/boto_resumable_upload.py) for resumable uploads.
- Retrying requests in Storage Transfer Service (/storage-transfer/docs/create-client#retry) with Java or Python.
- Dealing with JSON API upload errors using exponential backoff (/storage/docs/resumable-uploads#practices).

- Cloud Storage using exponential backoff to send object change notifications (/storage/docs/object-change-notification#_Reliable_Delivery) to notification subscribers.

Examples of backoff implemented in client libraries that you can use with Cloud Storage include:

- Retrying library (https://pypi.org/project/tenacity/) for Python.
- Google Cloud Client Libraries for Node.js (/nodejs/docs/reference/storage/latest/) can automatically use backoff strategies to retry requests with the `autoRetry` parameter.